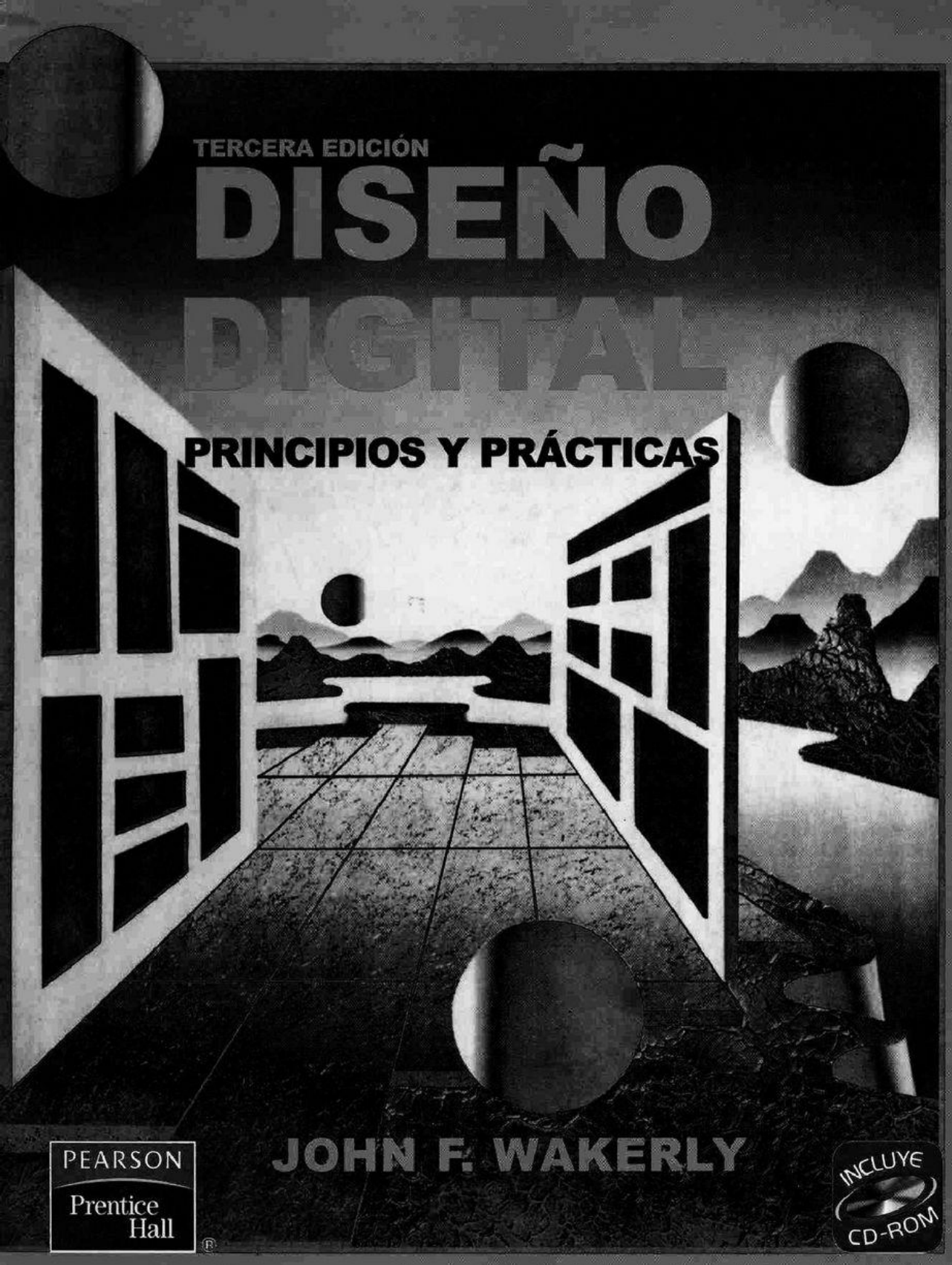
TERCERA EDICIÓN

DISEÑO DIGITAL

PRINCIPIOS Y PRÁCTICAS



JOHN F. WAKERLY

PEARSON
Prentice Hall

INCLUYE CD-ROM

DISEÑO DIGITAL
Principios y prácticas

Tercera edición

John F. Wakerly

Cisco Systems, Inc.
Stanford University

TRADUCCIÓN:
Efrén Alatorre Miguel
Ingeniero Físico

REVISIÓN TÉCNICA:
Oscar Yáñez Suárez
Profesor titular
Departamento de Ingenieria Eléctrica
Universidad Autónoma Metropolitana
Plantel Iztapalapa

Gonzalo Duchén Sánchez
Profesor Investigador
Sección de Estudios de Posgrado e Investigación
Escuela Superior de Ingeniería Mecánica y Eléctrica
Instituto Politécnico Nacional
Unidad Culhuacán

México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

Datos de catalogación bibliográfica

WAKERLY, JOHN F.
Diseño digital: principios y prácticas

PEARSON EDUCACIÓN, México, 2001

ISBN: 970-17-0404-5
Área: Universitarios

Formato: 18.5 × 23.5 cm Páginas: 976

Versión en español de la obra titulada *Digital design: principles and practices, third edition* de John F. Wakerly, publicada originalmente en inglés por Prentice Hall Inc., Upper Saddle River, New Jersey, U.S.A.

Esta edición en español es la única autorizada.

Original English language title by
Prentice Hall Inc.
Copyright © 2000
All rights reserved
ISBN 0-13-769191-2

Edición en español:
Editor: José Tomás Pérez Bonilla
Editor de desarrollo: Felipe de Jesús Castro Pérez
Supervisor de producción: Enrique Trejo Hernández

Edición en inglés:
Publisher: *Tom Robbins*
Editor-in-chief: *Marcia Horton*
Production editor: *Irwin Zucker*
Executive managing editor: *Vice O'Brien*
Assistant managing editor: *Eileen Clark*
Manufacturing buyer: *Pat Brown*
Copy editor: *Robert Lentz*
Director of production and manufacturing: *David W. Riccardi*
Art director: *Ann France*
Associate creative director: *Amy Rosen*
Cover art: *Robert McFadden*
Cover designer: *John Christiana*
Interior designer: *Donna Wickes*
Editorial assistant: *Dan Depasquale*

PRIMERA EDICIÓN, 2001

D.R. © 2001 por Pearson Educación de México, S.A. de C.V.
Atlacomulco Núm. 500-5° Piso
Col. Industrial Atoto
53519, Naucalpan de Juárez, Edo. de México

Cámara Nacional de la Industria Editorial Mexicana.
Registro No. 1031

Reservados todos los derechos. Ni la totalidad ni parte de esta publicación pueden reproducirse, registrarse o transmitirse, por un sistema de recuperación de información, en ninguna forma ni por ningún medio, sea electrónico, mecánico, fotoquímico, magnético o electroóptico, por fotocopia, grabación o cualquier otro, sin permiso previo por escrito del editor.

El préstamo, alquiler o cualquier otra forma de cesión de uso de este ejemplar requerirá también la autorización del editor o de sus representantes.

ISBN: 970-17-0404-5

Impreso en México, *Printed in Mexico*

1 2 3 4 5 6 7 8 9 0 – 03 02 01

A mis maestros

CONTENIDO

PRÓLOGO xix
PREFACIO xxi

1 INTRODUCCIÓN 1
- 1.1 Acerca del diseño digital 1
- 1.2 Analógico contra Digital 3
- 1.3 Dispositivos digitales 6
- 1.4 Aspectos electrónicos del diseño digital 7
- 1.5 Aspectos de software del diseño digital 9
- 1.6 Circuitos integrados 12
- 1.7 Dispositivos de lógica programable 15
- 1.8 Circuitos integrados (CI) de propósitos o aplicaciones específicas 16
- 1.9 Tarjeta de circuito impreso 18
- 1.10 Niveles del diseño digital 19
- 1.11 El objetivo del juego 22
- 1.12 Un paso adelante 23
- Problemas propuestos 24

2 SISTEMAS Y CÓDIGOS NUMÉRICOS 25
- 2.1 Sistemas numéricos posicionales 26
- 2.2 Números octales y hexadecimales 27
- 2.3 Conversiones generales de sistema numérico posicional 29
- 2.4 Suma y resta de números no decimales 32
- 2.5 Representación de números negativos 34
  - *2.5.1 Representación de magnitud con signo*
  - *2.5.2 Sistemas numéricos de complemento*
  - *2.5.3 Representación de complemento de base*
  - *2.5.4 Representación de complemento a dos*
  - *2.5.5 Representación de complemento de base reducida*
  - *2.5.6 Representación de complemento a unos*
  - *2.5.7 Representaciones por exceso*

2.6 Suma y resta de complemento a dos 39
2.6.1 Reglas de la suma
2.6.2 Una visión gráfica
2.6.3 Desbordamiento
2.6.4 Reglas de la resta
2.6.5 Números binarios sin signo y complemento a dos
2.7 Suma y resta de complemento a uno 44
2.8 Multiplicación binaria 45
2.9 División binaria 47
2.10 Códigos binarios para números decimales 48
2.11 Código Gray 51
2.12 Códigos de carácter 53
2.13 Códigos para acciones, condiciones y estados 53
2.14 Cubos n y distancia 57
2.15 Códigos para detectar y corregir errores 58
2.15.1 Códigos de detección de error
2.15.2 Corrección de errores y códigos de detección de errores múltiples
2.15.3 Códigos de Hamming
2.15.4 Códigos CRC
2.15.5 Códigos bidimensionales
2.15.6 Códigos de suma de verificación
2.15.7 Códigos m de n
2.16 Códigos para el almacenamiento y la transmisión de datos en serie 69
2.16.1 Datos en paralelo y en serie
2.16.2 Códigos de línea en serie
Referencias 73
Problemas propuestos 74
Ejercicios 76

3 CIRCUITOS DIGITALES 79
3.1 Compuertas y señales lógicas 80
3.2 Familias lógicas 84
3.3 Lógica CMOS 86
3.3.1 Niveles lógicos CMOS
3.3.2 Transistores MOS
3.3.3 Circuito del inversor básico CMOS
3.3.4 Compuertas CMOS NAND *y* NOR
3.3.5 Cargabilidad de entrada
3.3.6 Compuertas no inversoras
3.3.7 Compuertas CMOS AND-OR-INVERSORA *y* OR-AND-INVERSORA
3.4 Comportamiento eléctrico de los circuitos CMOS 96
3.4.1 Resumen
3.4.2 Hojas de datos y especificaciones
3.5 Comportamiento eléctrico de estado estable de los dispositivos CMOS 99
3.5.1 Niveles lógicos y márgenes de ruido
3.5.2 Comportamiento del circuito con cargas resistivas

3.5.3 Comportamiento del circuito con entradas no ideales
3.5.4 Fan out
3.5.5 Efectos de la carga
3.5.6 Entradas que no se utilizan
3.5.7 Picos de corriente y capacitores de desacoplamiento
3.5.8 Cómo destruir un dispositivo CMOS
3.6 Comportamiento eléctrico dinámico de los dispositivos CMOS 113
3.6.1 Tiempo de transición
3.6.2 Retardo de propagación
3.6.3 Consumo de energía
3.7 Otras estructuras CMOS de entrada y salida 123
3.7.1 Compuertas de transmisión
3.7.2 Entradas de un disparador Schmitt
3.7.3 Salidas de tres estados
3.7.4 Salidas de drenaje abierto
3.7.5 Control de LED
3.7.6 Buses de fuente múltiple
3.7.7 Lógica alambrada
3.7.8 Resistores de arranque
3.8 Familias lógicas CMOS 135
3.8.1 HC y HCT
3.8.2 VHC y VHCT
3.8.3 Características eléctricas de HC, HCT, VHC y VHCT
3.8.4 FCT y FCT-T
3.8.5 Características eléctricas del FCT-T
3.9 Lógica bipolar 145
3.9.1 Diodos
3.9.2 Lógica de diodos
3.9.3 Transistores bipolares de unión
3.9.4 Inversor en lógica de transistor
3.9.5 Transistores Schottky
3.10 Lógica transistor-transistor 156
3.10.1 Compuerta NAND TTL básica
3.10.2 Niveles lógicos y márgenes de ruido
3.10.3 Fan out
3.10.4 Entradas sin utilizar
3.10.5 Tipos adicionales de compuertas TTL
3.11 Familias TTL 166
3.11.1 Familias TTL iniciales
3.11.2 Familias TTL Schottky
3.11.3 Características de las familias TTL
3.11.4 Una hoja de datos TTL
3.12 Interfaz CMOS/TTL 170
3.13 Lógica CMOS de bajo voltaje e interfaz 171
3.13.1 Lógica LVCMOS y LVTTL de 3.3 V
3.13.2 Entradas tolerantes a 5 V
3.13.3 Salidas tolerantes a 5 V

3.13.4 Resumen de interfaces TTL/LVTTL
3.13.5 Lógica de 2.5 V y de 1.8 V
3.14 Lógica de emisor acoplado 175
3.14.1 Circuito básico CML
3.14.2 Familias ECL 10K/10H
3.14.3 Familia ECL 100K
3.14.4 ECL positivo (PECL)
Referencias 183
Problemas propuestos 184
Ejercicios 188

4 PRINCIPIOS DE DISEÑO LÓGICO-COMBINACIONAL 193
4.1 Álgebra de conmutación 194
4.1.1 Axiomas
4.1.2 Teoremas de una sola variable
4.1.3 Teoremas de dos y tres variables
4.1.4 Teoremas de n variables
4.1.5 Dualidad
4.1.6 Representaciones estándar de funciones lógicas
4.2 Análisis de circuitos combinacionales 209
4.3 Síntesis de circuitos combinacionales 215
4.3.1 Diseños y descripciones de circuito
4.3.2 Manipulaciones de circuito
4.3.3 Minimización del circuito combinacional
4.3.4 Mapas de Karnaugh
4.3.5 Minimización de sumas o productos
4.3.6 Simplificación de productos de sumas
4.3.7 Combinaciones de entrada "sin importancia"
4.3.8 Minimización de salida múltiple
4.4 Métodos de minimización programados 236
4.4.1 Representación de términos de producto
4.4.2 Cálculo de implicantes primos mediante la combinación de términos de producto
4.4.3 Encontrando una cubierta mínima empleando una tabla de implicantes primos
4.4.4 Otros métodos de minimización
4.5 Riesgos en el tiempo 244
4.5.1 Riesgos estáticos
4.5.2 Hallando riesgos estáticos mediante el uso de mapas
4.5.3 Riesgos dinámicos
4.5.4 Diseño de circuitos libres de riesgos
4.6 El lenguaje de descripción de hardware ABEL 249
4.6.1 Estructura del programa ABEL
4.6.2 Funcionamiento del compilador de ABEL
4.6.3 Instrucciones WHEN y bloques de ecuaciones
4.6.4 Tablas de verdad

4.6.5 Intervalos, conjuntos y relaciones
4.6.6 Entradas sin importancia
4.6.7 Vectores de prueba
4.7 El lenguaje de descripción de hardware VHDL 264
4.7.1 Flujo de diseño
4.7.2 Estructura de programa
4.7.3 Tipos y constantes
4.7.4 Funciones y procedimientos
4.7.5 Biblioteca y paquetes
4.7.6 Elementos de diseño estructural
4.7.7 Elementos de diseño de flujo de datos
4.7.8 Elementos de diseño basado en el comportamiento
4.7.9 La dimensión tiempo y simulación
4.7.10 Síntesis
Referencias 298
Problemas propuestos 301
Ejercicios 304

5 PRÁCTICAS DE DISEÑO LÓGICO COMBINACIONAL 311
5.1 Estándares de documentación 312
5.1.1 Diagramas de bloques
5.1.2 Símbolos de las compuertas
5.1.3 Nombres de señal y niveles activos
5.1.4 Niveles activos para terminales
5.1.5 Diseño de lógica burbuja-burbuja
5.1.6 Representación de la ubicación de los componentes
5.1.7 Buses
5.1.8 Información adicional del diagrama esquemático
5.2 Temporización del circuito 330
5.2.1 Diagramas de temporización
5.2.2 Retardo de propagación
5.2.3 Especificaciones de temporización
5.2.4 Análisis de temporización
5.2.5 Herramientas para análisis de temporización
5.3 PLD combinacionales 337
5.3.1 Arreglos de lógica programable
5.3.2 Dispositivos lógicos de arreglo programable
5.3.3 Dispositivos lógicos de arreglo genérico
5.3.4 Circuitos PLD bipolares
5.3.5 Circuitos PLD CMOS
5.3.6 Prueba y programación del dispositivo
5.4 Decodificadores 351
5.4.1 Decodificadores binarios
5.4.2 Símbolos lógicos para elementos de mayor escala
5.4.3 El decodificador de 2 a 4 dual 74x139
5.4.4 El decodificador de 3 a 8 74x138

5.4.5 Decodificadores binarios en cascada
5.4.6 Decodificadores en ABEL y en los PLD
5.4.7 Decodificadores en VHDL
5.4.8 Decodificadores de siete segmentos
5.5 Codificadores 376
5.5.1 Codificadores de prioridad
5.5.2 El codificador de prioridad 74x148
5.5.3 Codificadores en ABEL y PLD
5.5.4 Codificadores en VHDL
5.6 Dispositivos de tres estados 385
5.6.1 Buffers de tres estados
5.6.2 Buffers de tres estados SSI y MSI estándar
5.6.3 Salidas de tres estados en ABEL y PLD
5.6.4 Salidas de tres estados en VHDL
5.7 Multiplexores 398
5.7.1 Multiplexores MSI estándar
5.7.2 Expansión de multiplexores
5.7.3 Multiplexores, demultiplexores y buses
5.7.4 Multiplexores en ABEL y PLD
5.7.5 Multiplexores en VHDL
5.8 Compuertas OR Exclusivas y circuitos de paridad 410
5.8.1 Compuertas OR Exclusivas y NOR Exclusivas
5.8.2 Circuitos de paridad
5.8.3 El generador de paridad de 9 bits 74x280
5.8.4 Aplicaciones de verificación de paridad
5.8.5 Compuertas OR Exclusiva y circuitos de paridad en ABEL y PLD
5.8.6 Compuertas OR Exclusivo y circuitos de paridad en VHDL
5.9 Comparadores 419
5.9.1 Estructura del comparador
5.9.2 Circuitos iterativos
5.9.3 Un circuito comparador iterativo
5.9.4 Comparadores MSI estándar
5.9.5 Comparadores en ABEL y PLD
5.9.6 Comparadores en VHDL
5.10 Sumadores, restadores y ALU 430
5.10.1 Medio sumadores y sumadores completos
5.10.2 Sumadores iteractivos ("en rizo")
5.10.3 Sustractores (restadores)
5.10.4 Sumadores de acarreo anticipado
5.10.5 Sumadores MSI
5.10.6 Unidades lógico-aritméticas MSI (ALU)
5.10.7 Acarreo de grupo anticipado
5.10.8 Sumadores en ABEL y PLD
5.10.9 Sumadores en VHDL

5.11 Multiplicadores combinacionales 446
5.11.1 Estructuras de multiplicador combinacional
5.11.2 Multiplicación en ABEL y PLD
5.11.3 Multiplicación en VHDL
Referencias 455
Problemas propuestos 456
Ejercicios 459

6 EJEMPLOS DE DISEÑO DE CIRCUITOS COMBINACIONALES 467
6.1 Ejemplos de diseño de bloques de construcción 468
6.1.1 Desplazador rápido (Barrel shifter)
6.1.2 Codificador de punto flotante simple
6.1.3 Codificador de prioridad dual
6.1.4 Comparadores en cascada
6.1.5 Comparador dependiente del modo
6.2 Ejemplos de diseño usando ABEL y PLD 479
6.2.1 Desplazador rápido
6.2.2 Codificador de punto flotante simple
6.2.3 Codificador de prioridad dual
6.2.4 Comparadores en cascada
6.2.5 Comparador dependiente del modo
6.2.6 Contador de unos
6.2.7 Juego de "Tres en raya" o "Gato"
6.3 Ejemplos de diseño utilizando VHDL 500
6.3.1 Desplazador rápido
6.3.2 Codificador de punto flotante simple
6.3.3 Codificador de prioridad dual
6.3.4 Comparadores en cascada
6.3.5 Comparador dependiente del modo
6.3.6 Contador de unos
6.3.7 Juego de "Tres en raya" o "Gato"
Ejercicios 527

7 PRINCIPIOS DE DISEÑO LÓGICO SECUENCIAL 529
7.1 Elementos biestables 531
7.1.1 Análisis digital
7.1.2 Análisis analógico
7.1.3 Comportamiento metaestable
7.2 Latchs y flip-flops 534
7.2.1 Latch S-R
7.2.2 Latch $\overline{S}$-$\overline{R}$
7.2.3 Latch S-R con habilitación
7.2.4 Latch D
7.2.5 Flip-flop D disparado por flanco
7.2.6 Flip-flop D disparado por flanco con habilitación
7.2.7 Flip-flop de exploración

7.2.8 Flip-flop S-R maestro/esclavo
7.2.9 Flip-flop J-K maestro/esclavo
7.2.10 Flip-flop J-K disparado por flanco
7.2.11 Flip-Flop T
7.3 Análisis de una máquina de estado sincrónica temporizada 550
7.3.1 Estructura de la máquina de estado
7.3.2 Lógica de salida
7.3.3 Ecuaciones características
7.3.4 Análisis de máquinas de estado con flip-flops D
7.3.5 Análisis de máquinas de estado con flip-flops J-K
7.4 Diseño de máquina de estado sincrónica temporizada 563
7.4.1 Ejemplo de diseño de tabla de estado
7.4.2 Minimización de estado
7.4.3 Asignación de estado
7.4.4 Síntesis utilizando flip-flops D
7.4.5 Síntesis utilizando flip-flops J-K
7.4.6 Más ejemplos de diseño que utilizan flip-flops D
7.5 Diseño de máquinas de estado que utilizan diagramas de estado 584
7.6 Síntesis de una máquina de estado que utiliza listas de transición 591
7.6.1 Ecuaciones de transición
7.6.2 Ecuaciones de excitación
7.6.3 Variaciones sobre el esquema
7.6.4 Realización de la máquina de estado
7.7 Otro ejemplo de diseño de máquina de estado 594
7.7.1 El juego de las adivinanzas
7.7.2 Estados sin utilizar
7.7.3 Asignación de estado codificado por salida
7.7.4 Codificaciones de estado "sin importancia"
7.8 Descomposición de las máquinas de estado 602
7.9 Circuitos secuenciales de retroalimentación 604
7.9.1 Análisis
7.9.2 Análisis de circuitos con lazos de retroalimentación múltiple
7.9.3 Carreras
7.9.4 Tablas de estado y tablas de flujo
7.9.5 Análisis de flip-flop D CMOS
7.10 Diseño de circuito secuencial de retroalimentación 615
7.10.1 Latchs
7.10.2 Diseño de tabla de flujo de modo fundamental
7.10.3 Minimización de la tabla de flujo
7.10.4 Asignación de estado libre de carreras
7.10.5 Ecuaciones de excitación
7.10.6 Riesgos esenciales
7.10.7 Resumen
7.11 Características de diseño de circuito secuencial ABEL 627
7.11.1 Salidas registradas
7.11.2 Diagramas de estado

7.11.3 Memoria de estado externa
7.11.4 Especificación de salidas de Moore
7.11.5 Especificación de salidas Mealy y canalizadas con `WITH`
7.11.6 Vectores de prueba
7.12 Características de diseño de circuito secuencial VHDL 641
7.12.1 Circuitos secuenciales de retroalimentación
7.12.2 Circuitos temporizados
Referencias 644
Problemas propuestos 646
Ejercicios 650

8 PRÁCTICAS DE DISEÑO LÓGICO SECUENCIAL 659
8.1 Estándares de documentación de circuitos secuenciales 660
8.1.1 Requerimientos generales
8.1.2 Símbolos lógicos
8.1.3 Descripciones de máquina de estado
8.1.4 Especificaciones y diagramas de tiempo
8.2 Latchs y flip-flops 666
8.2.1 Latchs y flip-flops SSI
8.2.2 Inhibición del rebote de un interruptor
8.2.3 El inhibidor de rebote para el interruptor más simple
8.2.4 Circuito retenedor de bus
8.2.5 Registros y latchs de bits múltiples
8.2.6 Registros y latchs en ABEL y PLD
8.2.7 Registros y latchs en VHDL
8.3 PLD secuenciales 681
8.3.1 PLD secuenciales bipolares
8.3.2 Dispositivos GAL secuenciales
8.3.3 Especificaciones de temporización del PLD
8.4 Contadores 693
8.4.1 Contadores iterativos
8.4.2 Contadores sincrónicos
8.4.3 Contadores MSI y aplicaciones
8.4.4 Decodificación de estados de contador binario
8.4.5 Contadores en ABEL y VHDL
8.4.6 Contadores en VHDL
8.5 Registros de corrimiento 712
8.5.1 Estructura del registro de corrimiento
8.5.2 Registros de corrimiento MSI
8.5.3 La más grande aplicación de registro de corrimiento del mundo
8.5.4 Conversión serie/paralelo
8.5.5 Contadores de registro de corrimiento
8.5.6 Contador en anillo
8.5.7 Contadores Johnson

8.5.8 Contadores de registro de corrimiento con retroalimentación lineal
8.5.9 Registros de corrimiento en ABEL y PLD
8.5.10 Registros de corrimiento en VHDL
8.6 Circuitos iterativos contra secuenciales 747
8.7 Metodología de diseño sincrónico 750
8.7.1 Estructura del sistema síncrono
8.7.2 Un ejemplo de diseño de sistema sincrónico
8.8 Impedimentos para el diseño sincrónico 757
8.8.1 Sesgo de reloj
8.8.2 Disparo del reloj
8.8.3 Entradas asíncronas
8.9 Falla del sincronizador y metaestabilidad 764
8.9.1 Falla de sincronizador
8.9.2 Tiempo de resolución de la metaestabilidad
8.9.3 Diseño de sincronizador confiable
8.9.4 Análisis de la temporización metaestable
8.9.5 Mejores sincronizadores
8.9.6 Otros diseños de sincronizador
8.9.7 Flip-flops metaestables endurecidos
8.9.8 Sincronización de transferencias de datos de alta velocidad
Referencias 784
Problemas propuestos 786
Ejercicios 788

9 EJEMPLOS DE DISEÑO DE CIRCUITOS SECUENCIALES 795
9.1 Ejemplos de diseño que utilizan ABEL y PLD 796
9.1.1 Temporización y encapsulamiento de máquinas de estado basadas en PLD
9.1.2 Varias máquinas simples
9.1.3 Las luces traseras del Ford Thunderbird
9.1.4 El juego de las adivinanzas
9.1.5 Reinvención de los controladores de semáforos
9.2 Ejemplos de diseño que utilizan VHDL 813
9.2.1 Ejemplos de varias máquinas simples
9.2.2 Las luces traseras del Thunderbird
9.2.3 El juego de las adivinanzas
9.2.4 Reinvención de los controladores de semáforos

10 MEMORIAS, DISPOSITIVOS CPLD Y FPGA 831
10.1 Memoria de sólo lectura 832
10.1.1 Uso de las memorias ROM para funciones lógicas combinacionales "aleatorias"
10.1.2 Estructura interna de la ROM
10.1.3 Decodificación bidimensional
10.1.4 Tipos comerciales de ROM
10.1.5 Temporización y entradas de control de ROM
10.1.6 Aplicaciones de ROM

10.2 Memoria de lectura/escritura 854
10.3 RAM estática 854
10.3.1 Entradas y salidas de la RAM estática
10.3.2 Estructura interna de la RAM estática
10.3.3 Temporización de RAM estática
10.3.4 RAM estáticas estándar
10.3.5 SRAM sincrónica
10.4 RAM dinámica 866
10.4.1 Estructura de una memoria dinámica tipo RAM
10.4.2 Temporización de la RAM dinámica
10.4.3 DRAM sincrónicas
10.5 Dispositivos lógicos programables complejos 872
10.5.1 La familia CPLD XC9500 de Xilinx
10.5.2 Arquitectura de bloque de función
10.5.3 Arquitectura de bloque de entrada/salida
10.5.4 Matriz de conmutación
10.6 Arreglos de compuertas programable por campo 882
10.6.1 La familia FPGA XC4000 de Xilinx
10.6.2 Bloque de lógica configurable
10.6.3 Bloque de entrada/salida
10.6.4 Interconexión programable
Ejercicios 892

11 TEMAS ADICIONALES DEL MUNDO REAL 895
11.1 Herramientas de diseño asistido por computadora 895
11.1.1 Lenguajes de descripción de hardware
11.1.2 Captura de diagramas esquemáticos
11.1.3 Especificaciones y diagramas de temporización
11.1.4 Análisis y simulación del circuito
11.1.5 Asignación de componentes en la tarjeta de circuito impreso
11.2 Diseño para pruebas 902
11.2.1 Pruebas
11.2.2 Dispositivos de prueba y equipos de prueba en el circuito
11.2.3 Métodos de exploración
11.3 Estimación de la confiabilidad del sistema digital 908
11.3.1 Índice de fallas
11.3.2 Confiabilidad y tiempo medio entre fallas
11.3.3 Confiabilidad del sistema
11.4 Líneas de transmisión, reflexiones y terminaciones 912
11.4.1 Teoría básica de la línea de transmisión
11.4.2 Interconexiones de señal lógica como líneas de transmisión
11.4.3 Terminaciones de señal lógica
Referencias 920

ÍNDICE 923

PRÓLOGO

La Ley de Moore, que establece que la tecnología de los semiconductores avanza de manera exponencial, ha mantenido su validez por más de tres décadas. Los expertos predicen que seguirá así durante otro decenio. Cuando se introdujeron por primera vez los circuitos integrados, los paquetes lógicos contenían una docena o una cantidad similar de transistores. En la actualidad, debido a incrementos exponenciales en la densidad de los circuitos, los chips de microprocesadores han rebasado la barrera de los 10 millones de transistores. En menos de otra década alcanzarán los 100 millones de transistores por chip.

Para seguir el ritmo de la Ley de Moore, las técnicas de diseño han cambiado drásticamente. Hubo una época en la que los circuitos lógicos fabricados a mano fueron la norma. En la actualidad los diseñadores desarrollan los circuitos a partir de descripciones de alto nivel. Las conexiones hacia las tarjetas de circuito impreso se realizan en el interior de los chips. Con la lógica programable, las funciones lógicas y las conexiones en el chip pueden actualizarse dentro del ambiente del usuario.

¿Cómo mantener al corriente la educación con la Ley de Moore? ¿Qué podemos hacer para permitir que los estudiantes practiquen hoy sus habilidades y las adapten mañana a las nuevas generaciones de dispositivos? Éste es el reto que enfrentó John Wakerly cuando inició este trabajo.

Su enfoque es polifacético. Se basa en los principios fundamentales del diseño digital que no se modifican con la tecnología, como son la lógica combinacional, la lógica secuencial y las máquinas de estado. El autor une estrechamente estos principios con herramientas y técnicas prácticas que enseñan cómo diseñar para la tecnología actual. Éstas incluyen cómo utilizar los lenguajes de diseño ABEL y VHDL, cómo estructurar diseños con grandes bloques de construcción y cómo instrumentar diseños con dispositivos de lógica programable. Estas técnicas son esenciales para obtener éxito en el diseño.

La meta más difícil es ayudar al estudiante para que se adapte a los cambios inevitables que están por venir. Para lograr lo anterior el autor revela lo que ocurre bajo la lógica. Por ejemplo, proporciona modelos de transistores para compuertas y los utiliza para exponer aspectos que están relacionados con temporización y ruido. Las compuertas pueden llegar a ser más rápidas, más densas y pueden utilizar diferentes voltajes de control, pero el hecho de asegurar un funcionamiento adecuado y confiable siempre será

una preocupación constante. Aprenderemos las características, restricciones y modos de falla y cómo diseñar para ellos. También aprenderemos mediante ejemplos con diseños alternativos, cómo juzgar la calidad del diseño y evaluar los compromisos. A medida que surgen nuevas tecnologías, seremos capaces de diseñar para ellas.

El autor mejora el enfoque con habilidades de presentación que son raras en los textos universitarios. El lector apreciará rápidamente la efectividad de las gráficas, el estilo ameno de la exposición y lo instructivo de los ejercicios.

La Ley de Moore condena a los libros de texto en este campo a tener una vida breve. No obstante, el texto de Wakerly es un clásico.

Harold S. Stone
Princeton, New Jersey

PREFACIO

El presente libro está dirigido a todas aquellas personas que desean diseñar y construir circuitos digitales reales. Está basado en la idea que, a fin de hacer esto, usted tiene que captar los fundamentos, pero al mismo tiempo necesita comprender cómo funcionan las cosas en el mundo real. Ésta es la razón por la cual se propone el tema de "principios y prácticas".

cursos de introducción

El material en este libro es adecuado para cursos de introducción de diseño lógico digital en ingeniería eléctrica o en computación, así como en ciencias de la computación. Los estudiantes de ciencias de la computación que no estén familiarizados con los conceptos básicos de electrónica o que sencillamente no están interesados en el comportamiento eléctrico de los dispositivos digitales pueden omitir el capítulo 3; el resto de la obra fue escrita con la idea que fuera independiente de este capítulo tanto como fuera posible. Por otro lado, *cualquier persona* que tenga los fundamentos básicos de electrónica y desee obtener un mayor conocimiento de electrónica digital, puede hacerlo leyendo el capítulo 3. Aquellos estudiantes *que no cuenten con* antecedentes de electrónica pueden obtener conocimientos básicos si consultan el texto "Electrical Circuits Review", de Bruce M. Fleischer, que es un tutorial electrónico de 20 páginas disponible en forma gratuita y en el sitio Web de este libro, `www.ddpp.com`.

conceptos básicos de electrónica

Aunque el nivel de este libro es introductorio, contiene mucho más material del que puede impartirse en un curso típico de introducción. Cuando comencé a escribirlo, encontré que había tantas cosas importantes por decir, que no cabrían en un curso de un trimestre de Stanford o en un libro de 400 páginas. Por consiguiente, he seguido mi práctica habitual de incluir por lo menos, *todo* lo que pienso que es medianamente importante y dejar que el profesor o el lector decidan ellos mismos lo que es más importante en un ambiente en particular. Para ayudar en la toma de estas decisiones, marqué con un asterisco los encabezados de las *secciones opcionales*. En general, estas secciones pueden omitirse sin riesgo de perder la continuidad en las secciones no opcionales posteriores.

secciones opcionales

Sin duda algunas personas optarán por utilizar este libro en cursos avanzados y de laboratorio. Los estudiantes avanzados querrán saltarse los fundamentos e ir directamente a los aspectos interesantes. Una vez que usted conozca los fundamentos, lo más importante e interesante en este libro radica en las secciones dedicadas a los lenguajes de

cursos avanzados y de laboratorio

aspectos interesantes

descripción de hardware ABEL y VHDL, donde usted descubrirá que sus cursos de programación en realidad le servirán como preparación previa al diseño de hardware.

diseñadores digitales

Este libro también se puede utilizar como referencia de estudio autodidacta para todos aquellos diseñadores digitales que trabajan a nivel profesional, quienes pueden pertenecer a cualesquiera de dos clases:

Principiante Si usted acaba de incorporarse a una empresa y se desempeña como diseñador de circuitos digitales, y tomó cursos de diseño que fueron muy "teóricos" en la universidad, debería concentrarse en los capítulos 3, 5, 6, y del 8 al 11 para prepararse para el mundo real.

Profesionista con experiencia Si usted tiene experiencia, tal vez no requiera todo el material de "prácticas" de este libro, sin embargo, los que incluyen principios de los capítulos 2, 4 y 7 le serán útiles para organizar sus pensamientos, y las discusiones que ahí se presentan sobre lo que se considera importante y lo que no lo es podrían mitigar su sentimiento de culpa por no haber utilizado un mapa de Karnaugh en 10 años. Los ejemplos de los capítulos 6, 8 y 9 le proporcionarían ideas adicionales y la apreciación ante una variedad de métodos de diseño. Por último, las descripciones de los lenguajes ABEL y VHDL, así como los ejemplos dispersos a través de los capítulos que van desde el 4 hasta el 9, pueden servirle como primera introducción organizada al diseño basado en HDL.

notas al margen

Todos los lectores deberían hacer buen uso del índice detallado y de las *notas al margen* que aparecen en el texto porque dirigen su atención hacia las definiciones y los temas importantes. Es probable que los temas que están resaltados en *esta* sección sean más marginales que importantes, pero quería mostrar mi sistema de formateo de texto.

Descripción de los capítulos

Lo que tenemos a continuación es una lista de descripciones breves de los once capítulos y de este libro. Esto puede recordarle la sección de las guías de software, titulada "Para aquellos que aborrecen leer manuales". Si usted lee esta lista, tal vez no tenga que leer el resto del libro.

- El capítulo 1 proporciona unas cuantas definiciones básicas y determina las reglas fundamentales para lo que pensamos que es o no es importante en este libro.
- El capítulo 2 es una introducción a los sistemas numéricos y códigos binarios. Aun los lectores que ya se encuentren familiarizados con los sistemas numéricos binarios por haber llevado algún curso de software, deberían leer las secciones 2.10 a la 2.13 para obtener una idea de la forma en la que el hardware utiliza los códigos binarios. Los estudiantes avanzados pueden conseguir una atractiva introducción a los códigos de detección de errores al leer las secciones 2.14 y 2.15. Todos deberían leer el material de la sección 2.16.1 que se emplea en algunos ejemplos de diseño en el capítulo 8.
- El capítulo 3 describe "todo lo que usted siempre quiso saber acerca de" la operación de un circuito digital, haciendo énfasis principal en las características eléctricas externas de los dispositivos lógicos. El punto de partida son los fundamentos básicos

de electrónica, incluyendo voltaje, corriente y Ley de Ohm. A todos los lectores que no están familiarizados con estos conceptos les recomendamos que consulten el libro "Electrical Circuits Review", que se mencionó anteriormente. Los lectores que no estén interesados en la operación de los circuitos reales, o que pueden darse el lujo de tener alguien más que les haga el trabajo sucio, pueden omitir este capítulo.

- El capítulo 4 enseña principios de diseño lógico combinacional, incluyendo álgebra de conmutación, o álgebra binaria y análisis de circuito combinacional, síntesis y minimización. Las introducciones a ABEL y VHDL aparecen al final de este capítulo.
- El capítulo 5 comienza con el estudio de los estándares para la documentación del sistema digital, probablemente la práctica más importante para los diseñadores que aspiran a comenzar su formación práctica. En este mismo capítulo se presentan los dispositivos de lógica programable (PLD), enfocándose en su capacidad de llevar a cabo funciones lógicas combinacionales. El resto del capítulo describe las funciones lógicas combinacionales más comúnmente utilizadas y sus aplicaciones. Para cada función, se describen los bloques estándar de construcción MSI, programas ABEL para realizaciones PLD y modelos VHDL.
- El capítulo 6 es una colección de ejemplos más extensos de diseño de circuitos combinacionales. En cada ejemplo, se muestra cómo puede efectuarse el diseño con bloques de construcción MSI (si es apropiado), ABEL y PLD, o VHDL que pueden ser dirigidos tanto a un CPLD como a un FPGA.
- El capítulo 7 enseña los principios de diseño lógico secuencial, comenzando con latchs y "flip-flops". El énfasis principal en este capítulo radica en el análisis y diseño de máquinas de estado sincrónicas en el tiempo. Sin embargo, para los audaces y valientes, este capítulo incluye una introducción a los circuitos de modo fundamental y el análisis y diseño de circuitos secuenciales de retroalimentación. El capítulo termina con secciones acerca de las características de ABEL y VHDL que soportan el diseño de los circuitos combinacionales.
- El capítulo 8 trata en su totalidad del diseño práctico de circuitos secuenciales. Como se hizo en el capítulo 5, este capítulo se enfoca en las funciones más comúnmente empleadas y proporciona ejemplos haciendo uso de bloques de construcción MSI, ABEL y PLO así como VHDL. Las secciones 8.8 y 8.9 discuten los impedimentos inevitables para el diseño ideal o completamente sincrónico y abordan el problema de cómo vivir sincrónicamente en un mundo asincrónico.
- El capítulo 9 es una colección de máquinas de estado y ejemplos de diseño de circuitos secuenciales más grandes. Cada ejemplo se realiza tanto usando ABEL para un PLD como empleando VHDL, el cual puede ser dirigido a un CPLD o FPGA.
- El capítulo 10 es una introducción a los dispositivos de memoria CPLD y FPGA. La cobertura de la memoria incluye la memoria sólo de lectura, así como las memorias de lectura-escritura estática y dinámica desde los puntos de vista de la circuitería interna y el comportamiento funcional. Las últimas dos secciones presentan la arquitectura CPLD y FPGA.

- En el capítulo 11 se discuten varios temas misceláneos del mundo real que son de interés para los diseñadores digitales. Cuando comencé a escribir lo que yo pensaba que sería un libro de 300 páginas, incluí este capítulo en la versión preliminar para reforzar el material "central" y cubrir más tópicos. Bueno, el libro es evidentemente lo suficientemente extenso sin él, pero este material es útil a pesar de todo.

La mayor parte de los capítulos contiene referencias, problemas propuestos y ejercicios. Los problemas propuestos son por lo regular preguntas de respuestas breves o preguntas que despiertan el interés en la práctica y que pueden responderse directamente con base en el material del texto, mientras que los ejercicios requieren una mayor reflexión de parte del alumno. Los problemas propuestos en el capítulo 3 son particularmente extensos y están diseñados para permitir que los tipos que no son eléctricamente borrables entren en el material.

Xilinx Foundation Tools (sólo para versión en inglés)

Xilinx, Inc. (San Jose, CA 95124) gentilmente nos ha permitido incluir sus herramientas de diseño digital "Foundation Express" en dos CD-ROM al final de este libro, sólo para las ediciones en inglés de Estados Unidos. Estas herramientas son bastante detalladas, incluyen un compilador ABEL, procesadores de lenguaje Verilog y VHDL, un paquete de dibujo esquemático y un simulador. Gran parte del software en este paquete se encuentra basado en las populares herramientas Active-CAD™ y Active-HDL™ de Aldec, Inc. El paquete incluye también el software FPGA Express™ de Sinopsis, que permite que los diseños en ABEL, VHDL y Verilog sean dirigidos a CPLD y FPGA. La versión incluida apoya ciertas partes Xilinx populares.

Aun cuando usted no esté listo para crear sus propios diseños originales, puede hacer uso de las herramientas Foundation para probar y modificar cualquiera de los ejemplos en el texto, puesto que el código fuente de todos ellos se encuentra disponible en el sitio Web del libro, el cual mencionaremos a continuación.

WWW.DDPP.COM

Se tienen disponibles materiales de apoyo para este libro en el propio sitio Web dedicado al libro, `www.ddpp.com`. Un recurso clave para los estudiantes es el conjunto de listados fuente para todos los programas de ejemplo C, ABEL y VHDL del libro. También se encuentran disponibles directorios del proyecto Foundation en formato ZIP (comprimido), que incluyen, además de los archivos fuente ABEL y VHDL, algunos esquemas que sirvan para utilizar y estimular algunos de los ejemplos de diseño.

Durante la preparación de esta edición, quedé sorprendido y encantado de ver cuánto material de referencia de diseño digital está disponible en la Web, especialmente de parte de los fabricantes de dispositivos. El sitio Web DDPP contiene una sección de referencias "vivas" con enlaces a muchos sitios útiles que usted puede emplear como un punto de partida para su propio estudio independiente.

Un par de apéndices de las ediciones anteriores están disponibles en el sitio Web ("Electrical Circuits Review" por Bruce M. Fleischer, y "IEEE Standard Symbols"). Los

estudiantes que toman cursos de laboratorio apreciarán las cuatro páginas que incluyen las guías de terminales de salida de circuitos integrados (CIs), que aparecen en las páginas de forros de las ediciones anteriores.

Una cosa que a los estudiantes les puede gustar o no gustar, es una nueva colección de ejercicios nuevos que espero que se incremente a medida que yo continúe enseñando diseño digital en Stanford y a medida que reciba contribuciones de otros colegas.

Cómo se preparó este libro

El texto de esta tercera edición fue convertido de la versión TEX de la segunda edición original en Adobe FrameMaker®. Las figuras que provienen de las ediciones anteriores fueron convertidas desde Cricket Draw en archivos EPS de Adobe Illustrator®.

Toda la escritura, edición, dibujos y diseño de circuitos fueron realizados en una PC ejecutando Windows 95 o 98 con 384 Mbytes de memoria, la cual, lamentablemente, todavía se bloqueaba cuando se abrían al mismo tiempo demasiados programas o archivos. Las buenas noticias son que este uso de la edición de herramientas y programas estándar, me ha permitido proporcionar a lectores y profesores una gran colección de materiales útiles en el sitio Web del libro, como se indicó anteriormente.

Errores

Advertencia: Este libro puede contener errores. El autor no asume ninguna responsabilidad por cualquier daño (ya sea fortuito, cerebral o de otra índole) que ocasionen los errores.

A propósito, esta advertencia haría felices a los abogados. Pero para hacerlo feliz *a usted* permítame asegurarle que se ha puesto un enorme cuidado en la preparación de este manuscrito para hacer que se encuentre tan libre de errores como sea posible. Estoy ansioso de aprender de los errores restantes de modo que puedan ser corregidos en futuras reimpresiones, ediciones y productos derivados. Por consiguiente, pagaré $5 para el primer descubridor de cada error no detectado, sea técnico, tipográfico o de otra naturaleza. Por favor envíe sus comentarios por correo electrónico utilizando el vínculo en el sitio Web, o escribiéndome a `john@wakerly.com`.

Cualquier lector puede obtener una lista actualizada de los errores descubiertos empleando el vínculo en el sitio Web. Será una transferencia de archivo muy breve, eso espero.

Agradecimientos

Muchas personas me ayudaron a hacer posible este libro. La mayoría de ellas me ayudaron en la edición anterior y se les agradece ahí. La preparación de la tercera edición ha sido una tarea más solitaria, pero fue más fácil gracias a mis colegas Mario Mazzola y Prem Jain de Cisco Systems. Ellos y la compañía hicieron posible que acortara mis compromisos en Cisco para laborar menos de medio tiempo durante los ocho meses que tomó preparar esta edición.

En cuanto a las ideas que atañen a los "principios" de este libro, todavía le debo un gran agradecimiento a mi maestro, consejero de investigación y amigo Ed McCluskey. Por el lado de las "prácticas", mi personal del "Salón de la Fama de los Diseñadores Digitales" que incluye (en orden cronológico) a: Ed Davidson, Jim McClure, Courtenay Heater, Sam Wood, Curt Widdoes, Prem Jain, Ted Tracy, Dave Raaum, Akhill Duggal, Des Young y Tom Edsall.

La semilla que me condujo a comenzar la escritura de este libro y muchos otros fue plantada a principios de la década 1970 por Harold Stone en Stanford. Me puso a trabajar examinando e indexando sus libros, y sus textos de organización de computadoras me inspiraron a escribir mi primera obra de software. ¡Ahora, me gustaría ofrecer a Harold mis gracias tardías por apoyarme en mis inicios y darle un agradecimiento especial por ayudarme a hacer ¡dos páginas más de esta edición!

En el verano de 1997, durante las etapas tempranas de la planeación de este libro, mi amigo y colega Jean-Pierre Steger tomó un año sabático de la Escuela de Ingeniería de Burgdorf, cerca de Berna, en Suiza, para ayudarme a dar el primer paso con VHDL, las herramientas Foundation de Xilinx y otros temas. Varias personas contribuyeron revisando comentarios y materiales para esta edición, incluyendo a John Birkner, Rebecca Farley, Don Gaubatz, John Gill, Linley Gwennap, Jesse Jenkins y Jeff Purnell.

Xilinx, Inc., naturalmente se merece el crédito por suministrar las herramientas Foundation que son un importante accesorio de la edición en inglés. Por el lado de las personas, el director original de su University Program, Jason Feinsmith fue de mucha ayuda y su recientemente nombrado director Patrick Kane ha dado su apoyo entusiasta a nuestros esfuerzos.

Desde que fue publicada la edición anterior, he recibido muchos comentarios amables de los lectores. Además de sugerir o motivar muchas mejoras, los lectores han señalado docenas de errores tipográficos y técnicos cuyas correcciones se han incorporado en esta tercera edición.

Mi editor patrocinador en Prentice Hall, Tom Robbins, es digno de un gran agradecimiento por su paciencia. Él es el segundo editor que fue atraído a Prentice Hall en parte por la (falsamente atractiva) perspectiva de trabajar en un proyecto conmigo, solamente para descubrir en el momento de su llegada que el proyecto estaba muy atrasado. Sin embargo, yo conocía a Tom desde principios de la década de 1980 cuando él intentó contratarme para un proyecto con otro editor y, desde entonces, habíamos tratado de encontrar una manera de trabajar juntos; ahora, por fin, estamos aquí, comenzando la tercera década de nuestra amistad.

El editor de producción Irwin Zucker también merece crédito por proporcionar una interfaz sumamente tranquila con el departamento de producción de la editorial y todas las horas de apoyo en las que me ayudó durante la etapa final del proyecto. Si no fuera por él, no podría salir esta mañana para tomar tres semanas de vacaciones que planee con tanta anticipación (¡lo que quiero decir es que si no hubiera terminado a tiempo, nuestro perro de 90 libras habría ido a Europa en mi lugar!). Robert Lenz también hizo un gran trabajo como editor del manuscrito con lo que yo podía vivir :-).

Le envío un especial agradecimiento al artista Robert McFadden, cuya imagen de portada se encuentra colgada en mi casa junto con varias otras de sus extravagantes obras. La pintura que encargué y él realizó durante un año, me proporcionó la motivación para abordar el *interior* del libro.

Parece como si algún desastre siempre surgiera cuando estoy por completar uno de estos proyectos. En la edición anterior, fue el terremoto de la Serie Mundial de 1989. Esta vez, parece que he esquivado la bala hasta este momento. En realidad, no estará completamente hecho hasta que haya terminado el índice, el cual prepararé en mi computadora portátil mientras recorra sobre rieles el territorio europeo durante las próximas semanas. ¡Esperemos que no la olvide en algún tren!

Como siempre, debo agradecer a mi esposa Kate por soportar las horas de desvelo, frustración, mal humor, preocupación y llamadas telefónicas de gente extraña que ocurren cuando estoy comprometido en un proyecto de publicación como éste. ¡Esperamos que ustedes disfruten comenzar con este libro tanto como nosotros disfrutamos terminarlo!

John F. Wakerly
Mountain View, California

XILINX Edición del estudiante

Acceso a las Herramientas ISE-Webpack de Xilinx.
Debido a la constante actualización del software, es posible que usted no pueda hacer uso total de los CD incluidos en este libro, sin embargo, puede aprovechar el acceso sin costo por internet que Xilinx Inc. líder en el mercado de componentes electrónicos para lógica programable, ofrece a las herramientas de diseño digital ISE-Webpack.

Estas herramientas incluyen procesadores de lenguaje VHDL o Verilog, un editor de programas esquemáticos, un editor gráfico de diagramas de transición de estados, un generador de bancos de prueba, y un simulador (ModelSim XE II de Model Technology, Inc.), entre otras.

Para acceder a los archivos de instalación de ISE-Webpack, siga los siguientes pasos:

1. Dirigir el navegador a la dirección http://www.xilinx.com/support/download.htm.
2. Seleccionar la opción **WebPack ISE / ModelSim XE**
3. Pulsar el botón **Order & Register** y proporcionar la información solicitada. El registro es indispensable, no tiene costo, pero requiere una dirección válida de correo electrónico.
4. Al concluir el registro, elegir la opción **Download ISE WebPack**
5. Descargar los archivos **Complete ISE WebPACK Software y Complete MXE Simulator**
6. Proceder con la instalación, primero de ISE y luego de MXE

Si requiere mayor información, envíe un correo con sus observaciones a
guillermo.trujano@pearsoned.com

Estos CD-ROM están diseñados para usarse con los productos educativos "Xilinx Design Series" de Prentice Hall. El software está diseñado para instalar por primera vez la versión para estudiante del software Xilinx Foundation Series Express F1.5 y también para modernizar el software Xilinx Foundation Series F1.3 y F1.4 a la versión F1.5, además de proporcionar otros materiales electrónicos que podrían ser útiles. Para ejecutar correctamente el módulo VHDL y el software de síntesis Verilog (el módulo Synopsys FPGA Express™ de *Xilinx Foundation Series*) se requiere un archivo "licence.dat". Si va a modernizar la versión 1.3 o la 1.4, acceda a www.university.xilinx.com/univ/xsefaql.htm, donde se explica la pequeña modificación que debe hacerse al archivo de licencia existente. Si se trata de una instalación nueva, el archivo "licence.dat" se puede obtener en la página de inicio de Xilinx Design Series utilizando el número de serie impreso en la etiqueta que viene en su paquete Xilinx Design Series de Prentice Hall.

Este documento contiene las instrucciones de instalación básicas. Las instrucciones completas vienen en el CD-ROM Design Environment 1 de 2, en el archivo D:\readme.wri. Dichas instrucciones estaban completas en el momento de imprimirse el documento. Se pueden obtener versiones actualizadas de las instrucciones en
www.university.xilinx.com/univ/xdsl.htm

Sección I _ Xilinx Foundation Series Express F1.5
Sección II _ Otras herramientas
   a. (Opcional) Xilinx CORE Generator 1.5
   b. (Opcional) Herramientas XESS para tarjetas XS40 o XS95
   c. (Opcional) Simulador Active-VHDL 3.1 de Aldec (Versión para estudiante)
   d. (Opcional) Simulador Verilog SILOSIII de SimuCad (Versión para demostración)

   Sección III _ Configuración del archivo de licencia

Sección IV _ (Opcional) Adobe Acrobat y hojas de datos
Para modernizar F1.3 o F1.4 a F1.5, inicie en la Sección I.A.
Para instalar en una computadora nueva, inicie en la Sección I.B.

Sección I _ Xilinx Foundation Series Express F1.5

Sección I.A _ Desinstalar F1.3 o F1.4

1. Desinstale Foundation Project Manager (Design Environment): haga clic en Inicio, Configuración, Panel de control, Agregar o quitar programas
2. Desinstale Foundation Design Implementation Tools.
3. Si la desinstalación no funciona, borre el directorio C:\Xilinx, junto con todos sus subdirectorios y archivos (suponiendo que fue en ese directorio donde instaló sus Design Implementation Tools F1.3 o F1.4).

Sección I.B _ Instalar F1.5

1. Inserte el Disco de Instalación 1 de 2 de Design Environment. Inicie la instalación: haga clic en Inicio, Ejecutar y seleccione D:\setup.exe. Cuando se abra la ventana "Xilinx Foundation F1.5 Setup", haga clic en "Siguiente" (Next) y luego siga las indicaciones. Su clave de CD (CD KEY) es: FXAA2743390. ¡No pierda este número!
2. Si está efectuando una modernización, acuda a www.university.xilinx.com/univ/xds1.htm, donde se proporcionan instrucciones completas. Nota: Podría ser conveniente excluir los dispositivos XC4000E y SpartanXL, los cuales ocupan más de 50 MB de espacio en disco. Si más adelante quiere añadir estos dispositivos, puede ejecutar setup.exe otra vez.
3. Inserte el Disco de Instalación 2 de 2, Documentación. Inicie la instalación: haga clic en Inicio, Ejecutar y seleccione D:\setup.exe. Cuando se abra la ventana "Xilinx Foundation F1.5 Documentation Setup", haga clic en "Siguiente" (Next) y siga las indicaciones. No se necesita la CD KEY en este paso.
4. Si desea ahorrar espacio en disco, puede excluir "Multimedia Foundation Demo" en el cuadro de diálogo Select Products to Install (Seleccione los productos a instalar).
5. En "Registry Settings Options" (Opciones de registro), no modifique ninguna de las opciones.
6. Al terminar, reinicie su computadora para que entren en vigor los parámetros de Registro.

Sección II _ Otros archivos de herramientas para F1.5

El disco 2 contiene otras herramientas.

En general, la instalación es similar a la de las herramientas fundamentales. Haga clic en el botón Inicio, luego en Ejecutar, y pase al directorio de la herramienta que desea instalar. Haga doble clic en el archivo "setup.exe" y siga las indicaciones.

Las prácticas de laboratorio del Practical Xilinx Designer están en el disco 2 en D:\XESS\XLabs\... Hay un subdirectorio para cada práctica.

La información acerca de los requisitos del sistema está en el CD-ROM Design Environment Disk, disco 1 de 2, en D:\readme.wri, así como en el paquete de software.

La guía de instalación completa viene en el disco 2, en D:\Finstall\fndinst.pdf. Es preciso instalar Acrobat Reader para leer este archivo. Acrobat Reader se puede instalar desde el directorio ACROREAD en el disco 2.

Sección III _ Archivo de licencia para F1.5

En el caso de la versión F1.5, se necesita el archivo "licence.dat" para ejecutar las herramientas de síntesis VHDL y Verilog Synopsys FPGA Express.

Si está modernizando una instalación Xilinx Foundation Student Series F1.3 o F1.4 a F1.5, no necesita modificar su sistema. Es suficiente con su archivo "licence.dat" actual. Si va a usar la herramienta Express, vea el párrafo siguiente.

Si va a instalar el software Xilinx Foundation Series Express F1.5 en una computadora nueva, o simplemente no obtuvo un archivo "licence.dat" antes, visite el sitio Web para obtener su archivo de licencia y lea las instrucciones que se dan en www.university.xilinx.com/univ/xds1.htm.

Sección IV _ (Opcional) Adobe Acrobat y hojas de datos

Hay varios documentos en formato Adobe Acrobat que vienen incluidos en estos CD-ROM y que pueden servirle como referencia. Si desea obtener una documentación más completa acerca del software Xilinx Foundation Series Express F1.5 y su instalación, puede encontrar la F1.5 Quick Start Guide, versión comercial, en D:\installdoc\fndinst.pdf.

Éstos son los documentos clave que vienen incluidos en "Disco 2 de 2. Documentación de Foundation":

D:\Applinx\Databook.pdf Libro de datos 1998 de Xilinx
D:\Applinx\coregen.pdf Libro de datos de soluciones fundamentales Xilinx

Otros documentos clave que pueden consultarse empleando el visor de documentos Dynatext en el directorio Xilinx Book:

Foundation Series Quickstart Guide (Guía de inicio rápido)
Foundation Series User Guide (Guía de usuario)
Verilog Reference Guide (Guía de referencia de Verilog)
VHDL Reference Guide (Guía de referencia de VHDL)

Si necesita instalar Acrobat Reader, ejecute D:\ACRORED\WIN32\Ar32e30.exe y siga las instrucciones para instalar el software Acrobat Reader.

Además, hay varios proyectos de laboratorio en la página de inicio del Xilinx University Program, en la sección "Presentation Materials and Lab Files".
Visite **www.university.xilinx.com**

Requisitos mínimos para PC:
Se recomienda procesador Pentium"

Windows 95 o NT 4.0

Se recomienda 48 MB de RAM y 350 MB de espacio desocupado en el disco duro (más 100 MB de espacio para intercambio) antes de la instalación.

Dependiendo de las opciones que escoja, la instalación "Base" de Foundation Series Express F1.5 consumirá entre 250 y 500 MB de espacio en el disco duro. Esto no incluye Active-HDL ni Silos III.

Tenga presente que todo el apoyo para el software Xilinx Student Edition está basado en Web. Acceda a **www.university.xilinx.com** si desea información sobre resolución de problemas o de otro tipo.

capítulo 1

Introducción

Bienvenido al mundo del diseño digital. Tal vez usted sea un estudiante en ciencias de la computación que sabe todo acerca de programación y software de computadoras, pero todavía se encuentre intentando averiguar cómo es que todo ese intrincado hardware puede funcionar. O quizás sea un estudiante de ingeniería eléctrica que ya sabe algo acerca de electrónica analógica y diseño de circuitos, pero no sería capaz de reconocer un bit aun cuando lo tuviera enfrente. No importa. Comenzando desde un nivel bastante fundamental, este libro le mostrará cómo diseñar circuitos y subsistemas digitales.

Le proporcionaremos los principios básicos que necesitará para averiguar las cosas, y también le daremos muchos ejemplos. Junto con esos principios, intentaremos transmitirle la intuición que requiere el diseño digital del mundo real, discutiendo consideraciones prácticas y actuales siempre que sea posible. Y yo, el autor, con frecuencia haré referencia a mi persona como "nosotros" con la esperanza de que el lector se sienta guiado y tenga la impresión de que estamos caminando juntos a través del proceso de aprendizaje.

1.1 Acerca del diseño digital

Algunos lo llaman "diseño lógico". Eso está bien, pero al fin de cuentas el objetivo del diseño es construir sistemas. Para ese fin, en este texto cubriremos mucho más que únicamente teoremas y ecuaciones lógicas.

Este libro estudia los principios y prácticas. La mayoría de los principios que presentamos continuarán siendo importantes en los años que están por venir; algunos podrán aplicarse en formas que no se han descubierto todavía. En cuanto a las

prácticas, éstas pueden ser un poco diferentes de lo que se presente aquí en el momento que usted comience a trabajar en su ramo, y ciertamente continuarán cambiando a lo largo de su vida profesional. De modo que debería tratar con el material de "prácticas" en este libro como una manera de reforzar los principios, y como una forma de aprender los métodos de diseño.

Uno de los objetivos del libro es presentar lo suficiente acerca de los principios básicos para que usted sepa qué es lo que ocurre cuando utiliza herramientas de software que hacen el trabajo por usted. Los mismos principios básicos pueden ayudarle a llegar a la raíz de los problemas cuando las herramientas se cruzan en su camino.

En el cuadro de esta página se enumeran varios puntos clave que deberían aprenderse a través del estudio con este texto. La mayor parte de estos temas probablemente no tengan sentido para usted en este momento, no obstante usted deberá regresar y revisarlos posteriormente.

El diseño digital es ingeniería, y la ingeniería significa "resolución de problemas". Mi experiencia indica que solamente un 5 a 10% del diseño digital conforma "el aspecto divertido": la parte creativa del diseño, el destello de inspiración, la invención de un nuevo enfoque. Todo lo que resta es precisamente "trabajo pesado". Cabe mencionar que en la actualidad el trabajo pesado es mucho más fácil de realizar ahora, que hace 20 o incluso 10 años, sin embargo no podrá dedicar el 100% o incluso el 50% de su tiempo a trabajar en la parte divertida.

TEMAS IMPORTANTES EN EL DISEÑO DIGITAL

- El uso de buenas herramientas no garantiza un buen diseño, pero ayuda bastante a eliminar el esfuerzo y dolor que conlleva el hacer bien las cosas.
- Los circuitos digitales tienen características analógicas.
- Saber cuándo preocuparse y cuándo no preocuparse con respecto a los aspectos analógicos del diseño digital.
- Documentar siempre los diseños para hacerlos comprensibles tanto para uno mismo como para los demás.
- Asociar niveles activos con nombres de señal y practicar el diseño de lógica burbuja-burbuja.
- Comprender y hacer uso de bloques estándar de construcción funcional.
- Diseñar para un costo mínimo al nivel del sistema, incluyendo su propio esfuerzo ingenieril como parte del costo.
- El diseño de máquina de estado es como la programación: abórdelo desde este punto de vista.
- Haga uso de lógica programable para simplificar diseños, reducir costos y adaptar modificaciones de último momento.
- Evite el diseño asincrónico. Practique el diseño sincrónico hasta la llegada de una nueva tecnología.
- Localice las inevitables interfaces asincrónicas entre diferentes subsistemas y el mundo exterior, y suministre sincronizadores confiables.
- Encontrar un "glitch" o interferencia a tiempo evita otras nueve.

Además de la parte divertida y el trabajo pesado, existen otras muchas áreas en las cuales un diseñador digital con éxito debe ser competente, incluyendo las siguientes:

- *Depuración.* Es casi imposible ser un buen diseñador sin ser un buen localizador de fallas. La depuración acertada implica planeación, un enfoque sistemático, paciencia y lógica: ¡si no puede descubrir dónde *está* un problema, averígüe dónde *no se encuentra*!
- *Requerimientos y prácticas de negocios.* El trabajo de un profesional del diseño digital se ve afectado por un sinnúmero de factores externos a la ingeniería, incluyendo estándares de documentación, disponibilidad de componentes, definiciones de características, especificaciones del objetivo, organización o programación de tareas, políticas de oficina y comidas con los vendedores.
- *Toma de riesgos.* Cuando se comienza un proyecto de diseño deben evaluarse cuidadosamente los riesgos contra las consecuencias y recompensas potenciales, en áreas que abarcan desde la selección de nuevos componentes (¿estarán disponibles cuando yo esté listo para construir el primer prototipo?) hasta compromisos de programas de trabajo (¿conservaré mi empleo aunque me retrase en el proyecto?).
- *Comunicación.* Con el tiempo, usted ofrecerá sus diseños con éxito a otros ingenieros, departamentos y clientes. Si no tiene buenas habilidades de comunicación, nunca completará esta etapa en forma satisfactoria. Tenga en mente que la comunicación no sólo incluye la transmisión, también incluye la recepción de información; ¡aprenda a escuchar!

En el resto de este capítulo, y a lo largo del texto, continuaré expresando algunas opiniones acerca de lo que es importante y lo que no lo es. Pienso que tengo derecho a hacerlo siendo yo mismo un practicante de éxito moderado del diseño digital. Naturalmente, siempre serán bienvenidas sus opiniones y experiencias para compartir (para tal fin puede enviar un correo electrónico a la siguiente dirección: `john@wakerly.com`

1.2 Analógico contra Digital

analógicos

digitales

0

1

Los sistemas y dispositivos *analógicos* procesan las señales variantes en el tiempo que pueden adquirir cualquier valor a lo largo de un intervalo continuo de voltaje, corriente u otra medida. Del mismo modo lo hacen los sistemas y circuitos *digitales*: ¡la diferencia es que podemos aparentar que no lo hacen! Una señal digital se modela para tomar, en cualquier instante, solamente uno de dos valores discretos, que denominamos *0* y *1* (o BAJO y ALTO, FALSO y VERDADERO, negación y afirmación, Samuel y Pedro, o cosas por el estilo).

Las computadoras digitales han estado presentes desde los años cuarenta y se ha extendido su uso comercial desde los sesenta. Apenas en estos últimos 10 a 20 años la "revolución digital" se ha extendido a muchos otros aspectos de la vida. Ejemplos de aquellos sistemas analógicos que ahora se "han vuelto digitales" incluyen los siguientes:

- *Fotografías.* La mayoría de las cámaras todavía hacen uso de películas que tienen un recubrimiento de haluros de plata para grabar imágenes. Sin embargo, el incremento en la densidad de los microcircuitos o "chips" de memoria digital ha permitido el desarrollo de cámaras digitales que graban una imagen como una matriz de 640×480, o incluso arreglos más extensos de pixeles donde cada pixel almacena

las intensidades de sus componentes de color rojo, verde y azul, de 8 bits cada uno. Esta gran cantidad de datos, alrededor de siete millones de bits en este ejemplo, puede ser procesada y comprimida en un formato denominado JPEG y reducirse a un tamaño tan pequeño como el equivalente al 5% del tamaño original de almacenamiento, dependiendo de la cantidad de detalle de la imagen. De este modo, las cámaras digitales dependen tanto del almacenamiento como del procesamiento digital.

- *Grabaciones de video.* Un disco versátil digital, de múltiples usos (DVD, por las siglas de *digital versatile disc*) almacena video en un formato digital altamente comprimido denominado MPEG-2. Este estándar codifica una pequeña fracción de los cuadros individuales de video en un formato comprimido semejante al JPEG, y codifica cada uno de los otros cuadros como la diferencia entre éste y el anterior. La capacidad de un DVD de una sola capa y un solo lado es de aproximadamente 35 mil millones de bits, suficiente para grabar casi 2 horas de video de alta calidad, y un disco de doble capa y doble lado tiene cuatro veces esta capacidad.
- *Grabaciones de audio.* Alguna vez se fabricaron exclusivamente mediante la impresión de formas de onda analógicas sobre cinta magnética o un acetato (LP), las grabaciones de audio utilizan en la actualidad de manera ordinaria discos compactos digitales (CD, *Compact Discs*). Un CD almacena la música como una serie de números de 16 bits que corresponden a muestras de la forma de onda analógica original, se realiza una muestra por canal estereofónico cada 22.7 microsegundos. Una grabación en CD a toda su capacidad (73 minutos) contiene hasta seis mil millones de bits de información.
- *Carburadores de automóviles.* Alguna vez controlados estrictamente por conexiones mecánicas (incluyendo dispositivos mecánicos "analógicos" inteligentes que monitorean la temperatura, presión, etc.), en la actualidad los motores de los automóviles están controlados por microprocesadores integrados. Diversos sensores electrónicos y electromecánicos convierten las condiciones de la máquina en números que el microprocesador puede examinar para determinar cómo controlar el flujo de gasolina y oxígeno hacia el motor. La salida del microprocesador es una serie de números variante en el tiempo, que activa a transductores electromecánicos que, a su vez, controlan la máquina.
- *El sistema telefónico.* Comenzó hace un siglo con micrófonos y receptores analógicos que se conectaban en los extremos de un par de alambres de cobre (o, ¿ era una cuerda?). Incluso en la actualidad, en la mayor parte de los hogares todavía se emplean teléfonos analógicos, los cuales transmiten señales analógicas hacia la oficina central (CO) de la compañía telefónica. No obstante, en la mayoría de las oficinas centrales, estas señales analógicas se convierten a un formato digital antes que sean enviadas a sus destinos, ya sea que se encuentren en la misma oficina central o en cualquier punto del planeta. Durante muchos años los sistemas telefónicos de conmutación privados (PBX, private branch exchanges) que se utilizan en los negocios han transportado el formato digital todo el camino hacia los escritorios. En la actualidad, muchos negocios, oficinas centrales y los proveedores tradicionales de servicios telefónicos, están cambiando a sistemas integrados que combinan la voz digital con el tráfico digital de datos sobre una sola red de Protocolo de Internet IP (por las siglas en inglés de Internet Protocol).
- *Semáforos.* Para controlar los semáforos se utilizaban temporizadores electromecánicos que habilitaban la luz verde para cada una de las direcciones de circulación durante un intervalo predeterminado de tiempo. Posteriormente, se utilizaron rele-

vadores en módulos controladores que podían activar los semáforos de acuerdo con el patrón del tráfico detectado mediante sensores que se incrustan en el pavimento. Los controladores de hoy en día hacen uso de microprocesadores y pueden controlar los semáforos de modo que maximicen el flujo vehicular o, como sucede en algunas ciudades de California, sean un motivo de frustración para los automovilistas en un sinnúmero de creativas maneras.

- *Efectos cinematográficos.* Los efectos especiales creados exclusivamente para ser utilizados con modelos miniaturizados de arcilla, escenas de acción, trucos de fotografía y numerosos traslapes de película cuadro por cuadro. En la actualidad, naves espaciales, insectos, otras escenas mundanas e incluso bebés (en la producción animada de Pixar, *Tin Toy*) se sintetizan por completo haciendo uso de computadoras digitales. ¿Podrán algún día ya no ser necesarios ni los dobles cinematográficos femeninos o masculinos?

La revolución electrónica ha estado vigente bastante tiempo; la revolución del "estado sólido" comenzó con dispositivos analógicos y aplicaciones como los transistores y los radios transistorizados. Cabe preguntarse ¿por qué ha surgido ahora una revolución *digital*? De hecho, existen muchas razones para dar preferencia a los circuitos digitales sobre los circuitos analógicos:

- *Reproducibilidad de resultados.* Dado el mismo conjunto de entradas (tanto en valor como en serie de tiempo), cualquier circuito digital que hubiera sido diseñado en la forma adecuada, siempre producirá exactamente los mismos resultados. Las salidas de un circuito analógico varían con la temperatura, el voltaje de la fuente de alimentación, la antigüedad de los componentes y otros factores.
- *Facilidad de diseño.* El diseño digital, a menudo denominado "diseño lógico", es lógico. No se necesitan habilidades matemáticas especiales, y el comportamiento de los pequeños circuitos lógicos puede visualizarse mentalmente sin tener alguna idea especial acerca del funcionamiento de capacitores, transistores u otros dispositivos que requieren del cálculo para modelarse.
- *Flexibilidad y funcionalidad.* Una vez que un problema se ha reducido a su forma digital, podrá resolverse utilizando un conjunto de pasos lógicos en el espacio y el tiempo. Por ejemplo, se puede diseñar un circuito digital que mezcle o codifique su voz grabada de manera que sea absolutamente indescifrable para cualquiera que no tenga su "clave" (contraseña), pero ésta podrá ser escuchada virtualmente sin distorsión por cualquier persona que posea la clave. Intente hacer lo mismo con un circuito analógico.
- *Programabilidad.* Usted probablemente ya esté familiarizado con las computadoras digitales y la facilidad con la que se puede diseñar, escribir y depurar programas para las mismas. Pues bien, ¿adivine qué? Una gran parte del diseño digital se lleva a cabo en la actualidad al escribir programas, también, en los *lenguajes de descripción de hardware* (*HDLs*, por sus siglas en inglés). Estos lenguajes le permiten especificar o *modelar* tanto la estructura como la función de un circuito digital. Además de incluir un compilador, un HDL típico también tiene programas de simulación y síntesis. Estas herramientas de programación (software) se utilizan para verificar el comportamiento del modelo de hardware antes que sea construido, para posteriormente realizar la síntesis del modelo en un circuito, aplicando una tecnología de componente en particular.

lenguaje de descripción de hardware (HDL)

modelo de hardware

- *Velocidad.* Los dispositivos digitales de la actualidad son muy veloces. Los transistores individuales en los circuitos integrados más rápidos pueden conmutarse

PEQUEÑOS INTERVALOS DE TIEMPO

Un *microsegundo* (μseg) es igual a 10^{-6} segundos. Un *nanosegundo* (ns) equivale a 10^{-9} segundos y un *picosegundo* (ps) representa 10^{-12} segundos. En el vacío, la luz recorre aproximadamente una longitud de 1 pie en un nanosegundo, y una pulgada en 85 picosegundos. Con los transistores individuales en los circuitos integrados más rápidos, en los que la conmutación o cambio de estado se realiza en menos de 10 picosegundos, el retardo de la velocidad de la luz entre estos transistores a lo largo de un chip de silicio de media pulgada cuadrada ha llegado a ser un factor limitante en el diseño de circuitos.

en menos de 10 picosegundos, un dispositivo completo y complejo construido a partir de estos transistores puede examinar sus entradas y producir una salida en menos de 2 nanosegundos. Esto significa que un dispositivo de esta naturaleza puede producir 500 millones o más resultados por segundo.

- *Economía.* Los circuitos digitales pueden proporcionar mucha funcionalidad en un espacio pequeño. Los circuitos que se emplean de manera repetitiva pueden "integrarse" en un solo "chip" y fabricarse en masa a un costo muy bajo, haciendo posible la fabricación de productos desechables como son las calculadoras, relojes digitales y tarjetas musicales de felicitación. (Usted podría preguntarse, "¿acaso tales cosas son algo bueno?" ¡No importa!)
- *Avance tecnológico constante.* Cuando se diseña un sistema digital, casi siempre se sabe que habrá una tecnología más rápida, más económica o en todo caso una tecnología superior para éste en poco tiempo. Los diseñadores inteligentes pueden adaptar estos avances futuros durante el diseño inicial de un sistema, para anticiparse a la obsolescencia del sistema y para ofrecer un valor agregado a los consumidores. Por ejemplo, las computadoras portátiles a menudo tienen ranuras de expansión para adaptar procesadores más rápidos o memorias más grandes que las que se encuentran disponibles en el momento de su presentación en el mercado.

De este modo, esto es suficiente para un matiz de mercadotecnia acerca del diseño digital. El resto de este capítulo proporcionará una base un poco más técnica para prepararlo con miras al resto del libro.

1.3 Dispositivos digitales

compuerta

Los dispositivos digitales más elementales se conocen con el nombre de *compuertas* (del inglés *gates*, y no, no fueron precisamente bautizadas así en honor al fundador de cierta compañía prominente de software). Las compuertas originalmente deben su nombre a su capacidad de permitir o retardar (como lo haría una "compuerta") el flujo de la información digital. En general, una compuerta tiene una o más entradas y produce una salida que es una función del (los) valor(es) de la corriente de entrada. Mientras que las entradas y salidas pueden ser condiciones analógicas como voltaje, corriente, incluso presión hidráulica, se modelan de modo que tomen solamente dos valores discretos, 0 y 1.

compuerta AND

La figura 1-1 muestra los símbolos para las tres clases más importantes de compuertas. Una *compuerta AND* de 2 entradas, como la que muestra la parte (a) de la figura, produce una salida de 1 si sus dos entradas son iguales a 1; de otro modo produce una salida igual a 0. La figura muestra la misma compuerta cuatro veces, con las cuatro posibles

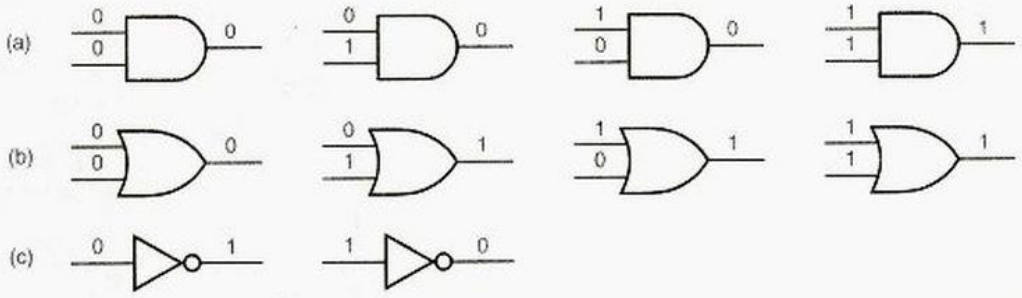


Figura 1-1 Dispositivos digitales: a) compuerta AND; b) compuerta OR; c) compuerta NOT o inversor.

combinaciones de entrada que pueden aplicarse a la misma y las salidas resultantes. Una compuerta también se conoce como *circuito combinacional* debido a que su salida depende solamente de la combinación de la corriente de entrada.

circuito combinacional

Una *compuerta OR* de 2 entradas, como la que se indica en la parte (b) de la figura 1-1, produce una salida 1 si una o más de sus entradas son 1; produce una salida 0 solamente si ambas entradas son 0. Una vez más, existen cuatro posibles combinaciones de entrada, cuyas salidas resultantes se ilustran en la figura.

compuerta OR

Una *compuerta NOT*, más comúnmente conocida como un *inversor*, produce un valor de salida que es el opuesto al valor de entrada, como se ilustra en la parte (c) de la figura.

compuerta NOT
inversor

Afirmamos que estas tres compuertas son las más importantes por una buena razón. Se puede realizar cualquier función digital haciendo uso sólo de estas tres clases de compuertas. En el capítulo 3 mostraremos cómo se realizan las compuertas empleando circuitos transistorizados. Sin embargo, usted debería saber que las compuertas han sido construidas o propuestas utilizando otras tecnologías, tales como relevadores, tubos de vacío, dispositivos hidráulicos y estructuras moleculares.

Un *flip-flop* (o biestable) es un dispositivo que almacena ya sea un 0 o un 1. El *estado* de un flip-flop es el valor que almacena actualmente. El valor almacenado puede modificarse solamente en ciertos instantes que están determinados por una entrada de "reloj", y el nuevo valor puede depender adicionalmente del estado lógico de la corriente del flip-flop y sus entradas de "control". Se puede construir un flip-flop a partir de una colección de compuertas que se conectan en forma inteligente, como lo demostraremos en la sección 7.2.

flip-flop
estado

Un circuito digital que contiene flip-flops se conoce como un *circuito secuencial*, porque su salida en cualquier momento depende no sólo de su corriente de entrada, sino también de la secuencia anterior de entradas que se han aplicado al mismo. En otras palabras, un circuito secuencial tiene *memoria* (recuerda) de los eventos pasados.

circuito secuencial

memoria

1.4 Aspectos electrónicos del diseño digital

Los circuitos digitales no son precisamente una versión binaria de sopa alfabética —con el debido respeto a la figura 1-1—, no tenemos ceros y unos pequeños flotando en torno a los circuitos. Como veremos en el capítulo 3, los circuitos digitales manejan voltajes y corrientes analógicos y se fabrican a partir de componentes analógicos. La "abstracción digital" permite que el comportamiento analógico sea ignorado en la mayoría de los casos, de manera que los circuitos puedan ser modelados como si en realidad procesaran ceros y unos.

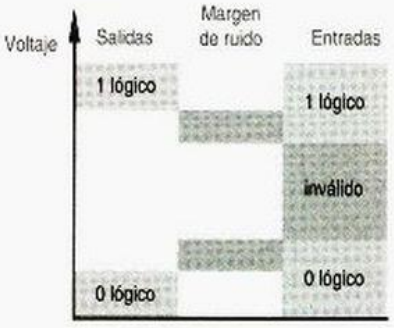


Figura 1-2
Valores lógicos y márgenes de ruido.

Un aspecto importante de la abstracción digital es la asociación de un *intervalo* de valores analógicos para cada valor lógico (0 o 1). Como se ilustra en la figura 1-2, no se garantiza que una compuerta típica tenga un nivel de voltaje preciso para una salida que corresponde al cero lógico. En lugar de ello, puede producir un voltaje dentro de un subintervalo del rango en el que se garantiza que va a ser reconocido como un 0 por las demás entradas de las compuertas. La diferencia entre las fronteras del intervalo se conoce como el *margen de ruido*: en un circuito real, la salida de una compuerta puede corromperse debido a este ruido excesivo, de cualquier forma su valor será interpretado correctamente en las entradas de otras compuertas.

margen de ruido

Este comportamiento es similar para las salidas que corresponden al 1 lógico. Nótese en la figura 1-2 que existe una región inválida entre los intervalos de entrada para el 0 lógico y el 1 lógico. Aunque cualquier dispositivo digital que funciona a un voltaje y temperatura particulares tendrá una frontera (o umbral) bastante bien definida entre ambos intervalos, otros dispositivos pueden tener diferentes fronteras. Más aún, todos los dispositivos que funcionan en forma adecuada tienen sus fronteras *en algún lugar* en el intervalo "inválido". Por tanto, cualquier señal que se encuentre dentro de los intervalos definidos para 0 y 1, será interpretada en forma idéntica por diferentes dispositivos. Esta característica es esencial para la reproducibilidad de los resultados.

La labor de un diseñador de circuitos *electrónicos* consiste en asegurarse que las compuertas lógicas produzcan y reconozcan las señales lógicas que se encuentran dentro de los intervalos apropiados. Éste es un problema de diseño de circuitos analógicos: abordaremos algunos de sus aspectos en el capítulo 3. No es posible diseñar un circuito que tenga el comportamiento deseado bajo cualquier condición de voltaje de alimentación, temperatura, carga y otros factores. En su lugar, el diseñador del circuito electrónico o fabricante del dispositivo proporciona *especificaciones* que definen las condiciones bajo las cuales se garantiza el comportamiento adecuado del dispositivo.

especificaciones

Por consiguiente, un diseñador *digital* no necesita hurgar en detalle en el comportamiento analógico de un dispositivo digital para verificar que funcione correctamente. En vez de ello, solamente necesita examinar el ambiente de operación del dispositivo para determinar que esté funcionando dentro de sus especificaciones. En efecto, se necesita cierto conocimiento analógico para realizar este examen, pero ni aún exagerando se compara con lo que necesitaría saber para diseñar un dispositivo digital partiendo desde cero. En el capítulo 3 daremos justamente lo que usted necesita.

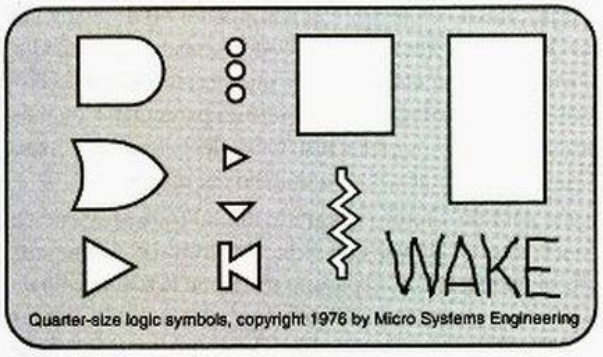


Figura 1-3
Una plantilla de diseño lógico.

1.5 Aspectos de software del diseño digital

El diseño digital tradicional no necesita involucrar a ninguna de las herramientas de software. Por ejemplo, la figura 1-3 muestra la herramienta principal de la "vieja escuela" del diseño digital: una plantilla de plástico que se utiliza para dibujar símbolos lógicos en diagramas esquemáticos hechos a mano (el nombre del diseñador se grababa en la plantilla con la punta de un cautín).

Sin embargo, en la actualidad, las herramientas de software son una parte esencial del diseño digital. En efecto, la disponibilidad y la factibilidad de los lenguajes de descripción de hardware (HDL, por sus siglas en inglés) y las herramientas de síntesis y simulación de circuitos que los acompañan han cambiado todo el panorama del diseño digital durante estos últimos años. Haremos un uso abundante de los lenguajes de descripción de hardware (HDL) a lo largo de este libro.

diseño asistido por computadora (CAD)

En el *diseño asistido por computadora* (*CAD*, por sus siglas en inglés) varias herramientas de software mejoran la productividad del diseñador y le ayudan a perfeccionar la exactitud y calidad de los diseños. En un mundo competitivo, el uso de herramientas de software es obligatorio para obtener resultados de alta calidad en programas de trabajo agresivos. A continuación se enumeran ejemplos importantes de herramientas de software para el diseño digital:

- *Captura del diagrama esquemático.* Para el diseñador digital, éste es el equivalente de un procesador de texto. Permite dibujar los diagramas esquemáticos "en línea", en lugar de hacerlo con papel y lápiz. Los programas de captura de diagramas esquemáticos más avanzados también verifican errores comunes, fáciles de localizar, como son: salidas en corto, señales que no van a ninguna parte, y así sucesivamente. Tales programas se discutirán con mayor detalle en la sección 11.1.2.
- *HDL.* Los lenguajes de descripción de hardware, originalmente fueron desarrollados para modelar circuitos, en la actualidad se utilizan cada vez más para el *diseño* de hardware. También se pueden utilizar para diseñar cualquier cosa, desde módulos de funciones individuales hasta grandes sistemas digitales con múltiples microcircuitos o chips. Presentaremos dos lenguajes de descripción de hardware ABEL y VHDL, al final del capítulo 4, y proporcionaremos ejemplos en ambos lenguajes en los siguientes capítulos.
- *Herramientas de síntesis, simuladores y compiladores HDL.* Un paquete típico de programas (software) HDL contiene varios componentes. En un ambiente típico, el

diseñador escribe un "programa" basado en texto, y el compilador HDL analiza el programa en busca de errores de sintaxis. Si se compila correctamente, el diseñador tiene la opción de entregarlo a una herramienta de síntesis que crea un diseño de circuito correspondiente enfocado a una tecnología de hardware en particular. Con más frecuencia, antes de la síntesis, el diseñador usará los resultados del compilador como la entrada de un "simulador" para verificar el comportamiento del diseño.

- *Simuladores.* El ciclo de diseño para un circuito integrado digital (que proviene de una oblea de silicio) es largo y costoso. Una vez que se ha construido el primer circuito integrado es muy difícil, si no imposible, depurarlo mediante la realización de pruebas en sus conexiones internas (éstas son realmente$_{\text{pequeñas}}$), o modificar las compuertas e interconexiones. Por lo regular, los cambios deben hacerse en la base de datos del diseño original, y por lo tanto, se debe fabricar un nuevo circuito integrado o "chip" para incorporar las modificaciones necesarias. Puesto que este proceso puede durar meses, los diseñadores de circuitos integrados están sumamente motivados para "hacerlo correctamente" (o casi correctamente) al primer intento. Los simuladores ayudan a los diseñadores a predecir el comportamiento eléctrico y funcional de un chip, sin tener que construirlo en realidad, no reconociendo así la mayor parte, si no es que todos los errores, antes de iniciar la fabricación del circuito.
- Los simuladores también se utilizan en el diseño de "dispositivos de lógica programable", que estudiaremos más adelante, y en el diseño total de sistemas que incorporan muchos componentes individuales. Estos últimos son menos críticos porque es más fácil para el diseñador efectuar cambios en los componentes e interconexiones de una tarjeta de circuito impreso. Sin embargo, aun un poco de simulación puede ahorrar tiempo al detectar errores simples pero estúpidos.
- *Bancos de prueba.* Los diseñadores digitales han aprendido cómo formalizar la prueba y simulación de circuitos en ambientes de software denominados "bancos de pruebas". La idea consiste en construir un conjunto de programas en torno a un diseño, para ejercitar de manera automática sus funciones y verificar su desempeño y el manejo de las señales en el tiempo. Esto es especialmente útil cuando se realizan cambios pequeños en el diseño: el banco de pruebas puede utilizarse para asegurar que las correcciones de los errores o "mejoras" que se aplican en un área no arruinen alguna otra. Los programas de los bancos de pruebas pueden escribirse en el mismo lenguaje de descripción de hardware que utiliza el diseño, en C o en C++, o en una combinación de lenguajes incluyendo lenguajes de "scripts" como PERL.
- *Verificadores y analizadores de temporización.* La dimensión de tiempo es muy importante en el diseño digital. Todos los circuitos digitales necesitan tiempo para producir un nuevo valor de salida en respuesta a un cambio en la entrada, y gran parte del esfuerzo del diseñador se emplea para asegurar que tales cambios en la salida sucedan con la rapidez suficiente (o, en algunos casos, no demasiado rápido). Los programas especializados pueden automatizar las tediosas tareas de dibujar diagramas de tiempo, especificar y verificar las relaciones de tiempo entre las diferentes señales que maneja un sistema complejo.
- *Procesadores de palabras.* No olvidemos el humilde editor de texto y el procesador de palabras. Estas herramientas obviamente son útiles en la creación del código fuente para los diseños basados en HDL, pero tienen una aplicación importante en todo diseño. ¡la creación de documentación!

DISPOSITIVOS DE LÓGICA PROGRAMABLE VERSUS SIMULACIÓN

En las siguientes secciones de este libro, usted encontrará que el uso de dispositivos de lógica programable (PLD) y arreglos de compuertas programable *in situ* (FPGA) le permitirán diseñar un circuito o subsistema al escribir cierta clase de programa. Los dispositivos PLD y FPGA que están disponibles en la actualidad contienen varios millones de compuertas, y las capacidades de estas tecnologías siguen creciendo. Si un diseño basado en FPGA o PLD no trabaja en la forma adecuada la primera vez, usted puede optar por realizar cambios en el programa y reprogramar físicamente el dispositivo, sin tener que cambiar algún componente o interconexión a nivel sistema. La facilidad de experimentación y modificación que tienen los dispositivos PLD y FPGA puede eliminar la necesidad de realizar simulaciones a nivel diseño de tarjeta; la simulación se requiere solamente en los diseños a nivel chip.

La visión que goza de mayor aceptación en las tendencias de la industria indica que conforme avanza la tecnología de fabricación de los chips, el diseño a nivel chip aumenta cada vez más, no así el diseño a nivel tarjeta. Por lo tanto, la capacidad para realizar una simulación completa y exacta adquiere una mayor importancia para el diseñador digital típico.

Sin embargo, existe otra visión. Si extrapolamos las tendencias en las capacidades de los dispositivos PLD y FPGA, en la próxima década seremos testigos del nacimiento de dispositivos que incluirán, además de compuertas, flip-flops y bloques de construcción, funciones de alto nivel como en el caso de los microprocesadores, memorias y controladores de entrada/salida. En este punto, la mayoría de los diseñadores digitales utilizarán componentes e interconexiones complejas que estarán integradas en los chips, cuyas funciones básicas habrán sido verificadas previamente por el fabricante de los dispositivos.

En esta visión del futuro, es posible que ciertas funciones programables de alto nivel no se apliquen correctamente, pero también existirá la posibilidad de arreglar los errores realizando un cambio sencillo en el programa; la simulación detallada del diseño antes de realizar el prototipo será una pérdida de tiempo. Otra visión compatible indica que los dispositivos PLD y FPGA serán simplemente un simulador de alta velocidad para el programa, ¡y este simulador será lo que se incluirá en el producto!

¿Acaso esta visión extrema tiene alguna validez? Para encontrar la respuesta, formúlese la siguiente pregunta: ¿Conoce algún programador (o programadores) de software que pueda depurar un programa nuevo con el simple hecho de "simular" su operación corriendo el programa?

En cualquier caso, los sistemas digitales modernos son bastante complejos como para permitir que el diseñador marque todas las condiciones posibles de entrada, con o sin la ayuda de las técnicas de simulación. Como sucede en el software, la operación adecuada de los sistemas digitales se logra aplicando prácticas que aseguren que los sistemas provienen de un "diseño sin errores". El propósito de esta obra es lograr este objetivo.

Además de utilizar las herramientas anteriores, los diseñadores en ocasiones pueden escribir programas especializados en lenguajes de alto nivel como C o C++, o scripts en lenguajes PERL, para resolver problemas particulares de diseño. Por ejemplo, la sección 10.1.6 nos proporciona un par de ejemplos de programas en C que generan las "tablas de verdad" para funciones lógicas combinacionales complejas.

Aunque las herramientas de CAD son importantes, por sí solas no hacen o deshacen al diseñador digital. Para tomar una analogía de otro campo, usted no se podría considerar un gran escritor solamente porque pueda mecanografiar rápidamente o porque sea muy hábil con un procesador de palabras. Durante sus estudios de diseño digital, asegúrese de aprender y utilizar todas las herramientas que estén a su disposición, tales como programas de edición de diagramas esquemáticos, simuladores y compiladores HDL. Pero recuerde que aprender a utilizar las herramientas no garantiza que usted sea capaz de producir buenos resultados. ¡Por favor, preste atención a lo que esté generando con ellas!

1.6 Circuitos integrados

circuito integrado (CI)

Una colección de una o más compuertas fabricadas en una oblea de silicio se conoce como *circuito integrado (CI)*. Los CI de gran tamaño que contienen decenas de millones de transistores pueden medir media pulgada o más por lado, mientras que los CI pequeños pueden medir menos de una décima de pulgada por lado.

oblea

Sin tener en cuenta su tamaño, un CI forma parte en un principio de una *oblea* circular mucho más grande, hasta de diez pulgadas de diámetro, que contiene docenas de cientos de réplicas del mismo CI. Todos los chips del CI en la oblea se fabrican al mismo tiempo como si fueran pizzas que se van a vender por rebanadas, excepto que en este caso, cada pedazo (chip de CI) se conoce como un *cuadro*. Después de que se fabrica la oblea, los cuadros se prueban directamente sobre la oblea y se marcan los que presentan defectos. Posteriormente la oblea se rebana para producir cuadros individuales y se descartan los que están marcados como defectuosos. (¡Compárese con el cocinero de pizzas que vende todas las rebanadas, incluso aquéllas sin suficiente pepperoni!) Cada cuadro sin marcar se monta en un encapsulado, y sus circuitos de entrada/salida se conectan a las terminales del encapsulado; por último, el CI encapsulado se somete a una prueba final y se envía al cliente.

cuadro

Algunas personas utilizan el término "CI" para referirse a un cuadro de silicio. Algunos utilizan el vocablo "chip" para referirse a la misma cosa. Incluso otros hacen uso de "CI" o "chip" para referirse a la combinación de un cuadro de silicio y su encapsulado. Los diseñadores digitales tienen la tendencia a emplear ambos términos en forma indistinta, y en realidad no les importa de lo que estén hablando. No requieren de una definición precisa, puesto que solamente toman en consideración el comportamiento eléctrico y funcional de estas cosas. En el resto de este libro, haremos uso del término *CI* para referirnos a un cuadro encapsulado.

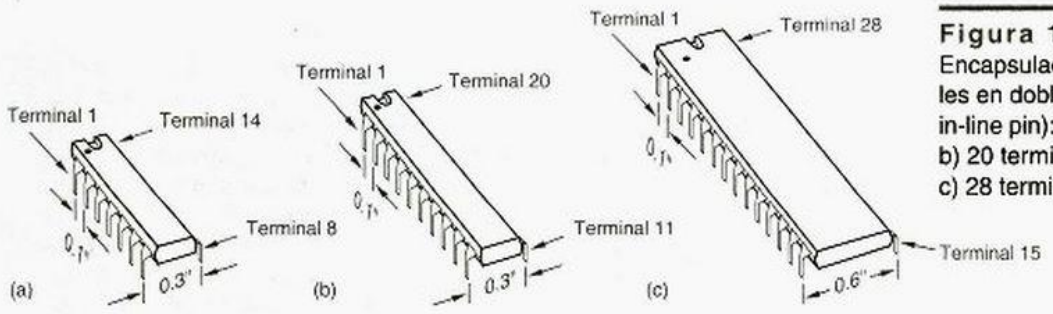


Figura 1-4
Encapsulados con terminales en doble línea (DIP, dual in-line pin): a) 14 terminales; b) 20 terminales; c) 28 terminales.

En los primeros días de los circuitos integrados, los CI se clasificaban por su tamaño (pequeños, medianos o grandes) de acuerdo con la cantidad de compuertas que contenían. El tipo más sencillo de CI que está disponible en el mercado todavía se conoce como circuito de *integración a pequeña escala (SSI, small-scale integration)* y contiene el equivalente de 1 a 20 compuertas. Los CI de integración a pequeña escala contiene por lo regular un puñado de compuertas o flip-flops, los bloques básicos de construcción del diseño digital.

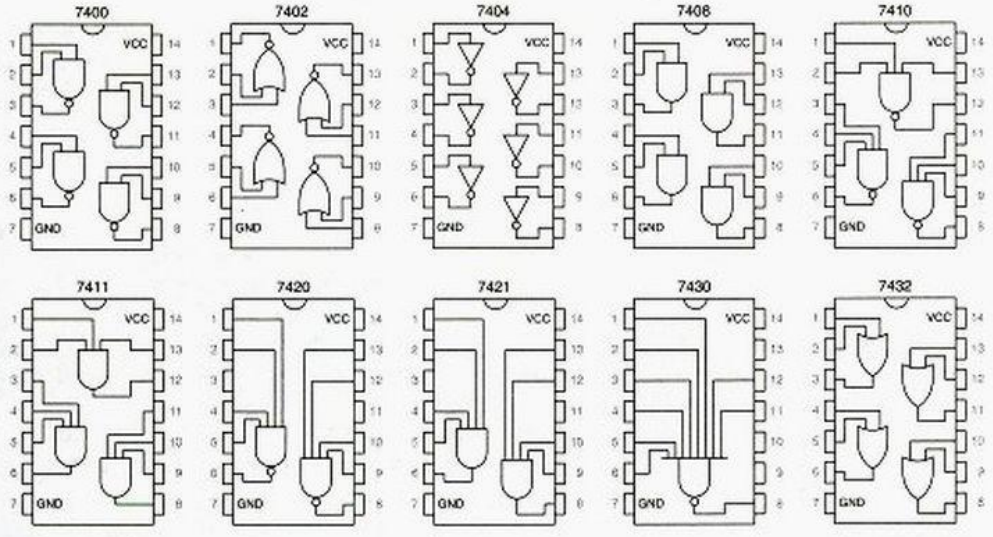
integración a pequeña escala (SSI)

Los CI de integración a pequeña escala que se utilizan en las prácticas de laboratorio (en los cursos de electrónica digital) vienen en un encapsulado de 14 terminales *en doble línea (DIP, dual in-line-pin)*. Como se ilustra en la figura 1-4(a), la separación entre las terminales en una columna es de 0.1 pulgadas y el espaciado entre columnas es de 0.3 pulgadas. Los encapsulados DIP más grandes incluyen funciones con más terminales, como se ilustra en (b) y en (c). El *diagrama de terminales* muestra la asignación de las señales del dispositivo con relación a las terminales del encapsulado, o *salidas*. La figura 1-5 muestra los diagramas de terminales para tipos comunes de circuitos integrados SSI. Tales diagramas se utilizan solamente para referencia mecánica, cuando un diseñador necesita determinar los números de terminales para un CI en particular. En el diagrama esquemático de un circuito digital, los

encapsulado de terminales en doble línea (DIP)

diagrama de terminales o salidas

Figura 1-5 Diagramas de terminales para varios CIs SSI de la serie 7400.



INTEGRACIÓN A ESCALA DIMINUTA

En los años venideros, quizás el uso más popular de los circuitos SSI y MSI, en especial en encapsulados DIP, será en los laboratorios de prácticas escolares. Estos dispositivos darán a los estudiantes la oportunidad de "utilizar sus manos" al ensamblar y conectar circuitos simples en un tablero de conexiones (*protoboard*) del mismo modo que sus profesores lo hicieron hace unos años.

Sin embargo, para mi gran sorpresa y deleite, un segmento de la industria de los CIs ha estado *disminuyendo* su escala hacia la integración a pequeña escala en los últimos años. La idea ha sido vender compuertas lógicas individuales en encapsulados muy pequeños. Estos dispositivos manejan funciones simples que en ocasiones son necesarias para ajustarse a componentes de mayor escala para un diseño particular, o en algunos casos se utilizan para trabajar en torno a errores en los componentes de mayor escala o sus interfaces.

Un ejemplo de un CI de esta clase es el CI 74VHC1G00 de Motorola. Este chip es una compuerta sencilla NAND de 2 entradas alojada en un encapsulado de 5 terminales (alimentación, tierra, dos entradas y una salida). El encapsulado completo, incluyendo las terminales, mide solamente ¡0.08 pulgadas de lado y tiene únicamente 0.04 pulgadas de altura! ¡Ahora esto es lo que llamaríamos "integración a escala diminuta"!

diagramas de terminales no se utilizan. En su lugar, las diversas compuertas se agrupan de manera funcional, como lo mostraremos en la sección 5.1.

Aunque los CI SSI todavía en ocasiones se utilizan como si fueran un "pegamento" que mantiene unidos a los elementos de mayor escala en sistemas complejos, han sido sustituidos en gran parte por dispositivos de lógica programable (PLD, programmable logic devices), los cuales estudiaremos en las secciones 5.3 y 8.3.

integración a mediana escala

Los siguientes CI que están disponibles en forma comercial se denominan circuitos de *integración a mediana escala (MSI)* y contienen el equivalente de aproximadamente 20 hasta 200 compuertas. En general, un CI de integración a mediana escala contiene un bloque de construcción funcional, como es el caso de un decodificador, registro o contador. En los capítulos 5 y 8 pondremos énfasis en estos bloques de construcción. Aun cuando el uso de CI discretos tipo MSI ha disminuido, los bloques de construcción equivalentes se emplean exhaustivamente en el diseño de CI de mayor tamaño.

integración a gra escala (LSI)

Los CIs de *integración a gran escala (LSI, large-scale integration)* son más grandes, contienen el equivalente de 200 a 200,000 compuertas o más. Las partes LSI incluyen memorias pequeñas, microprocesadores, dispositivos de lógica programable y dispositivos personalizados.

FUNCIONES LÓGICAS ESTÁNDAR

Muchas funciones estándar de "alto nivel" aparecen una y otra vez como los bloques de construcción del diseño digital. Históricamente, estas funciones fueron integradas primeramente en CI tipo MSI. Con el paso del tiempo, han surgido como componentes en las bibliotecas de "macros" para el diseño ASIC, como "celdas estándar" en diseño VLSI, como funciones "grabadas" en lenguajes de programación PLD y como funciones de bibliotecas en lenguajes de descripción de hardware tales como VHDL.

Las funciones lógicas estándar se presentan en los capítulos 5 y 8 como circuitos MSI de la serie 74, así como en forma HDL. La discusión y ejemplos en estos capítulos proporcionan una base para comprender y utilizar estas funciones.

integración a muy gran escala (VLSI)

La línea divisoria entre LSI y la *integración a muy gran escala (VLSI, very large-scale integration)* es difusa y tiende a definirse en términos de la cantidad de transistores en lugar de la cantidad de compuertas. Cualquier CI con más de 1,000,000 de transistores es en definitiva un VLSI, esto incluye a la mayor parte de los microprocesadores y memorias que se fabrican actualmente; además de los dispositivos de lógica programable de mayor tamaño y los dispositivos personalizados. En 1999, se estaban diseñando CIs de integración a muy gran escala con la cantidad de 50 millones de transistores.

1.7 Dispositivos de lógica programable

Existe una amplia variedad de CIs que pueden tener sus funciones lógicas "programadas" en su interior después de su fabricación. La mayor parte de estos dispositivos utilizan una tecnología que permite también su *re*programación, lo que significa que si usted encuentra un error en su diseño, puede ser capaz de arreglarlo sin reemplazar físicamente o volver a alambrar el dispositivo. En este libro, con frecuencia nos referiremos a los métodos y oportunidades de diseño para tales dispositivos.

arreglo de lógica programable (PLA)

Históricamente, los *arreglos de lógica programable (PLA, programmable logic arrays)* fueron los primeros dispositivos lógicos programables. Los PLA contenían compuertas AND y OR con una estructura de dos niveles, con conexiones programables por el usuario. Haciendo uso de esta estructura, un diseñador podía adaptar cualquier función lógica hasta un cierto nivel de complejidad utilizando la bien conocida teoría de síntesis lógica y minimización que presentaremos en el capítulo 4.

dispositivo lógico de arreglo programable (PAL)
dispositivo de lógica programable (PLD)

La estructura de los PLA sufrió mejoras y su costo disminuyó con la introducción de *dispositivos lógicos de arreglo programable (PAL)*. Actualmente, tales dispositivos se conocen en forma genérica como dispositivos de lógica programable (PLD, *programmable logic devices*) y son los "MSI" de la industria de la lógica programable. Tendremos mucho que decir acerca de la tecnología y arquitectura PLD en las secciones 5.3 y 8.3.

PLD complejo (CPLD)

La siempre creciente capacidad de los circuitos integrados creó una oportunidad para los fabricantes de CI para diseñar PLD más grandes para aplicaciones mayores de diseño digital. Sin embargo, por razones técnicas que discutiremos en la sección 10.5, la estructura básica de dos niveles AND-OR de los PLD no podía ser escalada a mayores dimensiones. En su lugar, los fabricantes de CIs idearon arquitecturas *complejas PLD (CPLD, complex PLD)* para conseguir la escala requerida. Un CPLD es simplemente una colección de varios PLD y una estructura de interconexión, todo en el mismo chip. Además de los PLD individuales, la estructura de interconexión del chip también es programable, lo que proporciona una amplia variedad de posibilidades de diseño. Los CPLD pueden ser escalados a tamaños más grandes si se aumenta la cantidad de PLD individuales y la complejidad de la estructura de interconexión en el chip CPLD.

arreglo de compuerta programable "in situ" (FPGA)

En forma paralela a la invención de los CPLD, otros fabricantes de CI tomaron un enfoque diferente para escalar el tamaño de los chips de lógica programable. En comparación con un CPLD, un arreglo de compuertas programable "in situ" (FPGA, *field-programmable gate arrays*) contiene una cantidad bastante superior de bloques lógicos individuales de menor tamaño, y proporciona una estructura de interconexión distribuida más grande que domina todo el chip. La figura 1-6 ilustra la diferencia entre ambos enfoques de diseño del chip.

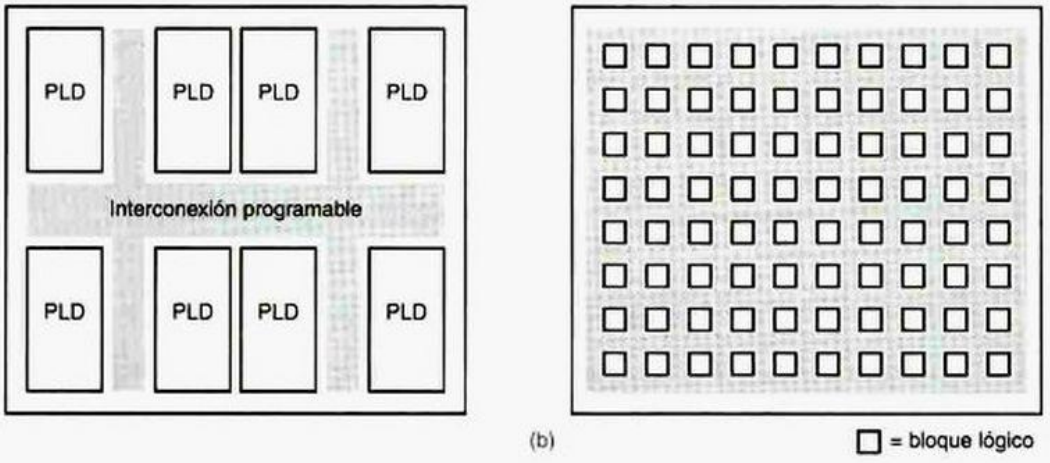


Figura 1-6 Enfoques de escalamiento para dispositivos de lógica programable de gran tamaño: a) CPLD; b) FPGA.

Los defensores de un enfoque o del otro usaban argumentos "religiosos" para ver cuál era mejor, pero el mayor fabricante de dispositivos lógicos programables, Xilinx, Inc., reconoció que había sitio para ambos enfoques; por lo tanto, inició la fabricación de las dos clases de dispositivos. Lo que es más importante que la arquitectura del chip es que los dos enfoques soportan un estilo de diseño en el cual los productos pueden ir desde el concepto del diseño hasta el prototipo y la producción en un tiempo muy corto.

El uso de los lenguajes de descripción de hardware es muy importante en la reducción del "tiempo de comercialización" para toda clase de productos basados en PLD. Los lenguajes como ABEL y VHDL, así como las herramientas de software que los acompañan, permiten que un diseño sea compilado, sintetizado y descargado en un PLD, CPLD o FPGA en pocos minutos. El poder de los lenguajes jerárquicos, altamente estructurados como VHDL, es especialmente importante para ayudar a los diseñadores a utilizar los cientos de miles o millones de compuertas que proporcionan los CPLD y FPGA de mayor tamaño.

1.8 Circuitos integrados (CI) de propósitos o aplicaciones específicas

Tal vez los descubrimientos más interesantes en la tecnología del CI para el diseñador digital promedio no son los siempre crecientes tamaños del chip, sino las siempre crecientes oportunidades para "diseñar su propio chip". Los chips diseñados para un producto o aplicación particular o limitada se denominan *CI semipersonalizados* o *CI de propósito específico (ASIC, application-specific IC)*. Los ASIC generalmente reducen los componentes totales y el costo de fabricación de un producto al reducir la cantidad de chips, el tamaño físico y el consumo de energía, y a menudo proporcionan un desempeño más elevado.

CI semipersonalizado
CI de aplicación específica (ASIC)

El *costo* de la *ingeniería no recurrente (NRE, nonrecurring engineering)* para el diseño de un ASIC puede exceder el de un diseño discreto, por $5,000 y hasta $250,000 USD o más. Los cargos de la NRE se reembolsan al fabricante del CI y a las empresas que son responsables de las siguientes actividades: diseño de la estructura interna del

costo de la ingeniería no recurrente (NRE)

chip, creación de las mascarillas de metal para fabricar los chips, desarrollo de pruebas para los chips, y fabricación de las primeras muestras del chip.

El costo típico de la NRE para un ASIC común, de complejidad media, que incluye cerca de 100,000 compuertas es de $30,000 a $50,000 USD. Normalmente, el diseño de un ASIC tiene sentido sólo cuando el costo de la NRE puede verse compensado por los ahorros por unidad sobre el volumen de venta esperado del producto.

LSI personalizado

El costo de la NRE para el diseño de un chip *personalizado* de integración a gran escala *LSI* (cuyas funciones, arquitectura interna y diseño detallado a nivel transistor, se ajustan a los requerimientos de un cliente específico) es muy elevado, puede ser igual o mayor a $250,000 USD. De este modo, el diseño LSI completamente personalizado se hace sólo para chips que tienen una aplicación comercial general o que disfrutarán de un volumen de ventas muy alto en una aplicación específica (por ejemplo, un chip de reloj digital, una interfaz de red o un circuito de interfaz de bus para una PC).

celdas estándar

Para reducir los costos de la NRE, los fabricantes de circuitos integrados han desarrollado conjuntos de programas de *celdas estándar* que incluyen las funciones de los circuitos MSI que se utilizan con mayor frecuencia, como son los decodificadores, registros y contadores; y las funciones de circuitos LSI que se utilizan con cierta frecuencia en las memorias, arreglos de lógica programable y microprocesadores. En un *diseño de celda estándar*, el diseñador lógico interconecta las funciones en forma similar, como en el caso de los chips múltiples MSI/LSI. Solamente se crean celdas personalizadas si es absolutamente necesario (con un costo adicional, por supuesto). Luego todas las celdas se depositan en el chip, se optimiza su ubicación para reducir los tiempos de retardo de propagación y minimizar las dimensiones del chip. Al minimizar el tamaño del chip se reduce el costo unitario del mismo, puesto que ello incrementa la cantidad de chips que se pueden fabricar en una oblea de silicio. El costo de la NRE para un diseño de una celda estándar es de $150,000 USD.

diseño de celda estándar

arreglo de compuertas

Si bien, muchas personas pueden considerar que $150,000 USD es una cantidad considerable, por lo tanto, los fabricantes de CIs han tomado un paso adelante al trasladar la capacidad del diseño ASIC a las masas. Un *arreglo de compuertas* es un CI cuya estructura interna está formada por un arreglo de compuertas y en un principio no se especifican sus interconexiones. El diseñador lógico especifica los tipos de compuertas e interconexiones. Aun cuando el diseño del chip se especifica en este nivel básico, por lo general el diseñador trabaja con "macroceldas", que son las mismas funciones de alto nivel que se utilizan en el diseño de chip múltiples MSI/LSI y en los diseños de celdas estándar; el software expande el diseño de alto nivel a uno de bajo nivel.

La diferencia principal que existe entre el diseño de celda estándar y el diseño de arreglo de compuertas es que las macroceldas y la disposición del chip de un arreglo de compuertas no están altamente optimizados como en el caso de un diseño de celda estándar, de modo que el tamaño del chip puede crecer un 25% o más, con el consiguiente aumento en el precio. Además, el enfoque de arreglo de compuertas no permite la creación de celdas personalizadas. Por otra parte, el diseño con arreglo de compuertas puede terminarse con mayor rapidez y a un costo menor de la NRE, abarcando desde los $5,000 (lo que se había dicho al principio) hasta $75,000 USD (el precio total cuando todo esté terminado).

Los métodos básicos de diseño digital que estudiaremos a lo largo de este libro se aplican muy bien al diseño funcional de ASIC. No obstante, existen oportunidades, restricciones y pasos adicionales en el diseño de ASIC, dependiendo del entorno de diseño y del proveedor particular de ASIC.

1.9 Tarjeta de circuito impreso

tarjeta de circuito impreso (PCB)

tarjeta de alambrado impreso (PWB)

Normalmente un circuito integrado se instala en una *tarjeta de circuito impreso (PCB, printed-circuit board)* [o *tarjeta de alambrado impreso (PWB, printed-wiring board)*] que lo conecta con otros circuitos integrados en un sistema. Las tarjetas de circuito impreso de capas múltiples (que se utilizan en los sistemas digitales típicos) tienen pistas de cobre que están grabadas sobre varias capas delgadas de fibra de vidrio, las cuales están laminadas formando una sola tarjeta que tiene un espesor cercano a 1/16 de pulgada.

pistas de la tarjeta

milésimas

línea fina

En general las conexiones individuales de alambrado, o *pistas de la tarjeta de circuito impreso*, son bastante angostas, normalmente tienen un espesor que va de 10 a 25 milésimas de pulgada. (Un *mil* significa la milésima parte de una pulgada.) En la tecnología de tarjetas de circuito impreso de *línea fina*, las pistas son extremadamente estrechas, miden 4 milésimas de ancho, y la separación entre las pistas adyacentes es de 4 milésimas de pulgada. De este modo, se pueden realizar hasta 125 conexiones en una franja de una pulgada de ancho, en uno de los lados de una tarjeta de circuito impreso. Si se necesita una densidad de conexiones más elevada, entonces pueden emplearse más capas.

tecnología de montaje superficial (SMT)

La mayor parte de los componentes que se instalan en las tarjetas modernas de circuito impreso utilizan la *tecnología de montaje superficial (SMT, surface-mount technology)*. En lugar de utilizar dispositivos con encapsulado DIP que tienen terminales largas, las cuales se hacen pasar a través de las perforaciones de la tarjeta y se sueldan en el lado inferior (lado de pistas) de la tarjeta, las terminales de los dispositivos que tienen encapsulados tipo SMT vienen dobladas para que hagan contacto con el lado superior de la tarjeta de circuito impreso. Antes de montar estos componentes en la tarjeta, se aplica una "pasta especial" para soldar en las almohadillas de contacto que están en la superficie de la tarjeta. Para esto se utiliza una plantilla cuyo patrón de orificios coincide con las almohadillas de contacto (de la tarjeta) donde se va a aplicar soldadura y la terminal del componente que se va a soldar. Después se colocan los componentes tipo SMT (ya sea en forma manual o con la ayuda de una máquina especial) sobre las almohadillas. Los componentes se mantienen en su sitio gracias a la pasta de soldar (o en algunos casos, mediante la aplicación de un adhesivo). Por último, el ensamble completo se hace pasar a través de un horno especial que funde la pasta de soldar, la cual se solidifica cuando se enfría.

El uso de la tecnología de componentes de montaje superficial, aunado a la tecnología de tarjetas de circuito impreso de línea fina, permite un empacamiento extremadamente denso de circuitos integrados y otros componentes sobre una tarjeta (PCB). Este empacamiento denso permite entre otras cosas, el ahorro de espacio. Para circuitos que operan a velocidades muy altas, esta densidad elevada de componentes permite la minimización de fenómenos analógicos adversos, incluyendo los efectos de las líneas de transmisión y las limitaciones inherentes a la velocidad de la luz.

módulo de chip múltiple (MCM)

Para satisfacer los requerimientos más rigurosos de velocidad y densidad, se han desarrollado *módulos de chips múltiples (MCM, multichip modules)*. En esta tecnología, el cuadro (oblea de silicio) que corresponde a un CI no se monta en un encapsulado individual de cerámica o plástico. Por el contrario, los cuadros de CI que integran un subsistema de alta velocidad (digamos, un microprocesador y su memoria caché) se unen directamente en un sustrato que contiene las interconexiones necesarias en forma de capas múltiples. El MCM se sella herméticamente y tiene sus propias terminales externas para alimentación, tierra y las entradas y salidas de señal que requiere el sistema que lo contiene.

1.10 Niveles del diseño digital

El diseño digital puede realizarse en diferentes niveles de representación y abstracción. Aunque se puede aprender y practicar el diseño a un nivel particular, de vez en cuando tendrá que subir o bajar un nivel o dos para efectuar el trabajo. También, la industria misma y la mayoría de los diseñadores se han desplazado a un ritmo constante hacia mayores niveles de abstracción a medida que la densidad y funcionalidad del circuito se han incrementado.

El nivel más bajo de diseño es la física del dispositivo y los procesos de fabricación del CI. Este nivel es responsable de los impresionantes avances en la densidad y velocidad del CI que han ocurrido en el transcurso de las décadas pasadas. Los efectos de estos avances están resumidos en la *Ley de Moore*, la cual fue establecida primero por el fundador de Intel Gordon Moore en 1965, e indica lo siguiente: la cantidad de transistores por pulgada cuadrada en un CI se duplica cada año. En años recientes, la proporción de avance ha disminuido para duplicarse aproximadamente cada 18 meses, pero es importante advertir que con cada duplicación en la densidad surge también una duplicación en la velocidad.

Ley de Moore

Este libro no desciende hasta el nivel de la física del dispositivo y procesos de fabricación del CI, pero usted necesita reconocer la importancia de ese nivel. Es importante estar consciente de probables avances tecnológicos y otros cambios en la planeación del producto y el sistema. Por ejemplo, la disminución en las dimensiones del chip (que se han logrado en tiempos recientes) han forzado a moverse a menores voltajes de alimentación lógicos, provocando mayores cambios en la manera en que los diseñadores planean y especifican sistemas modulares y actualizaciones.

En este libro, saltamos al diseño digital a nivel de transistores y recorremos todo el camino hasta el nivel del diseño lógico utilizando HDL. Nos detenemos cerca del siguiente nivel, que incluye el diseño de computadoras y el diseño de un sistema total. El "centro" de nuestro estudio se encuentra en el nivel de los bloques funcionales de construcción.

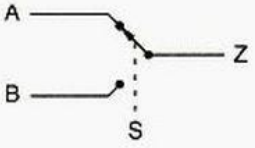


Figura 1-7
Modelo de interruptor para función multiplexora.

Para obtener un resumen previo de los niveles de diseño que cubriremos, considere un ejemplo sencillo de diseño. Supongamos que estamos por construir un "multiplexor" con dos bits de entradas de datos, A y B, un bit de entrada de control S y un bit de salida Z. Dependiendo del valor de S, 0 o 1, el circuito transfiere el valor de A (o bien de B) a la salida Z. Esta idea se ilustra en el "modelo de interruptor" de la figura 1-7. Consideremos el diseño de esta función en varios niveles diferentes.

Aunque en general el diseño lógico se realiza en un nivel más elevado, en el caso de algunas funciones se obtienen ventajas adicionales si se aplica el diseño al nivel transistor. El multiplexor es una función de esa clase. La figura 1-8 muestra cómo el multiplexor

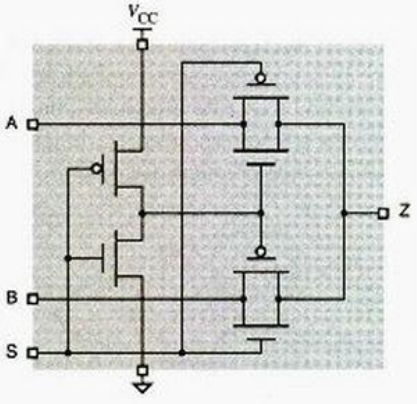


Figura 1-8
Diseño de un multiplexor utilizando compuertas de transmisión CMOS.

Tabla 1-1
Tabla de verdad para la función multiplexora.

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

puede diseñarse aplicando la tecnología "CMOS", la cual utiliza estructuras especializadas de circuitos transistorizados llamadas "compuertas de transmisión", que estudiaremos en la sección 3.7.1. Aplicando este enfoque, el multiplexor puede construirse con solamente seis transistores. Cualquiera de los otros enfoques que describimos requiere de por lo menos 14 transistores.

En el estudio tradicional del diseño lógico, utilizaríamos una "tabla de verdad" para describir la función lógica del multiplexor. Una tabla de verdad muestra todas las combinaciones posibles de valores de entrada y los correspondientes valores de salida para la función. Puesto que el multiplexor tiene tres entradas, tiene 2^3 o bien 8 posibles combinaciones de entrada, como se ilustra en la tabla 1-1.

Una vez que tenemos la tabla de verdad, los métodos de diseño lógico tradicionales, que se describen en la sección 4.3, utilizan el álgebra Booleana y los algoritmos de minimización para obtener una ecuación "optima" AND-OR de dos niveles a partir de la tabla de verdad. Para la tabla de verdad del multiplexor, tendríamos la siguiente ecuación:

$$Z = S' \cdot A + S \cdot B$$

Esta ecuación se lee "Z es igual a S complemento y A o S y B". Avanzando aun más, podemos convertir la ecuación en un conjunto correspondiente de compuertas lógicas que realizan la función lógica especificada, como se muestra en la figura 1-9. Este circuito requiere de 14 transistores si utilizamos la tecnología CMOS estándar para las cuatro compuertas mostradas.

El multiplexor es una función que se utiliza con bastante frecuencia, y la mayor parte de las tecnologías de lógica digital proporcionan bloques de construcción predefinidos de multiplexores. Por ejemplo, el CI 74x157 es un chip MSI que realiza la multiplexión en entradas de 4 bits en forma simultánea. La figura 1-10 corresponde a un diagrama lógico que muestra cómo podemos conectar precisamente un bit de este bloque de construcción de 4 bits para resolver el problema. Los números en color gris corresponden a la designación de las terminales de un encapsulado DIP de 16 terminales que contiene al dispositivo.

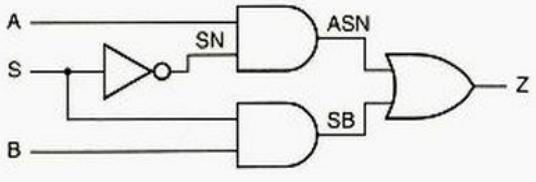


Figura 1-9
Diagrama lógico al nivel de compuerta para la función multiplexora.

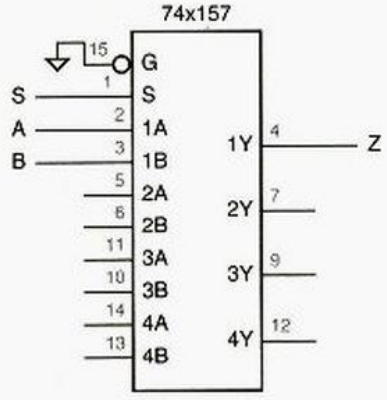


Figura 1-10
Diagrama lógico para un multiplexor utilizando un bloque de construcción MSI.

También podemos realizar la función multiplexora como parte de un dispositivo de lógica programable. Los lenguajes como ABEL nos permiten especificar salidas empleando ecuaciones booleanas semejantes a la de la página anterior, pero en general es más conveniente hacer uso de elementos de lenguaje de "alto nivel". Por ejemplo, la tabla 1-2 muestra un programa ABEL para la función multiplexora. Las primeras tres líneas definen el nombre del módulo del programa y especifican el tipo de PLD en el que la función se realizará. Las siguientes dos líneas especifican la designación de terminales del dispositivo para entradas y salida. La instrucción "WHEN" especifica la función lógica real en una manera que es muy fácil de comprender, aun cuando no hayamos estudiado ABEL todavía.

Un lenguaje de más alto nivel, VHDL, puede emplearse para especificar la función multiplexora en una forma que es muy flexible y jerárquica. La tabla 1-3 muestra el programa VHDL de ejemplo para el mutiplexor. Las primeras dos líneas especifican una "biblioteca" estándar y el conjunto de definiciones que se van a utilizar en el diseño. Las siguientes cuatro líneas especifican solamente las entradas y salidas de la función, ocultando a propósito cualquier detalle acerca de la manera en que la función se lleva a cabo de manera interna. La sección de "architecture" del programa especifica el comportamiento de la función. La sintaxis de VHDL requiere de cierta familiarización, pero la instrucción simple "when" nos dice básicamente lo mismo que nos dijo la versión en ABEL. En VHDL la "herramienta de síntesis" puede empezar con esta descripción de compor-

Tabla 1-2
Programa ABEL para el multiplexor.

```
module chap1mux
title 'Two-input multiplexer example'
CHAP1MUX device 'P16V8'

A, B, S      pin 1, 2, 3;
Z            pin 13 istype 'con';

equations

WHEN S == 0 THEN Z = A;  ELSE Z = B;

end chap1mux
```

Tabla 1-3
Programa VHDL para el multiplexor.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity  Vchap1mux is
    port ( A, B, S; in  STD_LOGIC;
           Z:       out STD_LOGIC );
end Vchap1mux

architecture Vchap1mux_arch of Vchap1mux is
begin
  Z <= A when S = '0' else B;
end Vchap1mux_arch;
```

tamiento y producir un circuito que tenga este comportamiento en una tecnología de lógica digital de objetivo específico.

Al forzar de manera explícita la separación de las definiciones de entrada/salida ("entidad") y la realización interna ("architecture"), VHDL facilita la definición de realizaciones alternas de funciones sin tener que efectuar cambios en otra parte de la jerarquía del diseño. Por ejemplo, un diseñador podría especificar una arquitectura alternativa y estructural para el multiplexor como se muestra en la tabla 1-4. Esta arquitectura es básicamente un equivalente de texto del diagrama lógico de la figura 1-9.

Avanzando un poco más, VHDL es bastante poderoso como para permitirnos definir en forma real las operaciones que modelan el comportamiento funcional a nivel transistor (aunque no exploraremos tales capacidades en este libro). De este modo, podríamos terminar el ciclo escribiendo un programa VHDL que especifique una realización en el ámbito del transistor del multiplexor equivalente de la figura 1-8.

Tabla 1-4
Programa VHDL "estructural" para el multiplexor.

```
architecture Vchap1mux_gate_arch of Vchap1mux_is
signal SN, ASN, SB: STD_LOGIC;
begin
  U1: INV (S, SN);
  U2: AND2 (A, SN, ASN);
  U3: AND2 (S, B, SB);
  U4: OR2 (ASN, SB, Z);
end Vchap1mux_gate_arch;
```

1.11 El objetivo del juego

diseño a nivel tarjeta de circuito impreso

Dados los requerimientos funcionales y de rendimiento para un sistema digital, el objetivo del juego en el diseño digital práctico es minimizar costos. Para *diseños a nivel tarjeta de circuito impreso* (sistemas que se ensamblan en una sola tarjeta de circuito impreso) esto implica minimizar el número de circuitos integrados. Si se requieren demasiados CIs, no cabrán todos en la tarjeta. "Bien, simplemente utilice una tarjeta más grande", diría usted. Por desgracia, las dimensiones de las tarjetas están restringidas por diversos factores, como es el caso de los estándares que existen en el mercado (p. ej. las

tarjetas de sonido, controladoras o de video para las computadoras personales); restricciones en las dimensiones del ensamble (p. ej. la tarjeta de control que tiene que caber en el interior del tostador); o disposiciones que marca la superioridad (p. ej. a fin de obtener la aprobación del proyecto hace tres meses, usted tontamente le indicó a su gerente que todo el diseño cabrá en una tarjeta de 3×5 pulgadas, ¡y ahora tiene que cumplirlo!). En cada uno de estos casos, el costo de usar una tarjeta más grande o varias tarjetas puede ser una opción inaceptable.

En general, la regla consiste en minimizar el número de circuitos integrados, aun cuando el costo del CI individual varíe. Por ejemplo, un CI típico de SSI o MSI puede costar 25 centavos (de dólar), mientras que un pequeño PLD puede tener un costo de un dólar. Es posible efectuar una función particular con tres CI SSI y MSI (75 centavos de dólar) o un PLD (un dólar). En la mayoría de los casos, se emplea la solución PLD de mayor precio, no porque el diseñador posea una acción en la compañía que fabrica los PLD, sino porque la solución PLD utiliza menos área de la tarjeta y también es mucho más fácil de modificar en caso que el diseño tenga fallas.

diseño ASIC

En el *diseño ASIC*, el objetivo del juego es un poco diferente, pero la importancia de las técnicas de diseño funcionales y estructuradas es la misma. Aunque es fácil gastar horas y semanas creando macroceldas personalizadas y minimizando el conteo total de compuertas de un ASIC, esto es aconsejable sólo en casos especiales. La reducción del costo unitario que se consigue al tener un chip 10% más pequeño es despreciable, excepto en aplicaciones de alto volumen. En aplicaciones con volumen bajo a volumen medio (la mayoría de ellas), existen dos factores de mayor importancia: tiempo de diseño y costo de la NRE.

Un tiempo más corto de diseño permite que el producto llegue al mercado con mayor prontitud, incrementando los ingresos sobre la vida media del producto. Un costo más bajo en la NRE también fluye directo a la "línea inferior" y en pequeñas compañías puede ser la única manera en la que el proyecto puede terminarse antes que se agote el presupuesto de la compañía (¡créamelo, yo he pasado por eso!). Si el producto tiene éxito, siempre resulta provechoso "pellizcar" el diseño para reducir los costos unitarios. La necesidad de minimizar el tiempo de diseño y el costo de la NRE es un argumento a favor de un enfoque estructurado al diseño ASIC, en oposición al enfoque altamente optimizado, utilizando bloques de construcción estándar que proporciona la biblioteca del fabricante de ASIC.

Las consideraciones en el diseño PLD, CPLD y FPGA son una combinación de lo anterior. En general, la selección de una tecnología PLD en particular y la selección de las dimensiones del dispositivo se realiza al principio del ciclo de diseño. Después, cuando el diseño se "ajusta" al dispositivo seleccionado, no vale la pena intentar optimizar el conteo de compuertas o el área de la tarjeta: el dispositivo ya se ha implementado. No obstante, si nuevas funciones o reparaciones de errores llevan al diseño más allá de la capacidad del dispositivo seleccionado, es entonces cuando se debe trabajar muy duro para modificar el diseño de manera que lo haga funcionar en la forma adecuada.

1.12 Un paso adelante

Con esta sección se termina el capítulo de introducción. A medida que continúe la lectura del libro, deberá recordar dos cosas. La primera, el principal objetivo del diseño digital es construir sistemas que resuelvan los problemas de las personas. Aun cuando este libro le proporcione las herramientas fundamentales para el diseño, como parte de

su trabajo usted deberá conservar la "imagen general del entorno de diseño" en su mente. En segundo lugar, el costo es un factor importante en toda decisión de diseño; recuerde que debe considerar no sólo el costo de los componentes digitales, sino también el de la actividad del diseño.

Finalmente, a medida que profundice más y más en el texto, si encuentra algo que crea haber visto antes pero no recuerda dónde, por favor consulte el índice. He intentado hacerlo tan útil y completo como me ha sido posible.

Problemas propuestos

1.1 Sugiera una imagen de mejor aspecto para colocarla en la primera página de la siguiente edición de este libro.

1.2 Proporcione tres diferentes definiciones para la palabra "bit" como se utiliza en este capítulo.

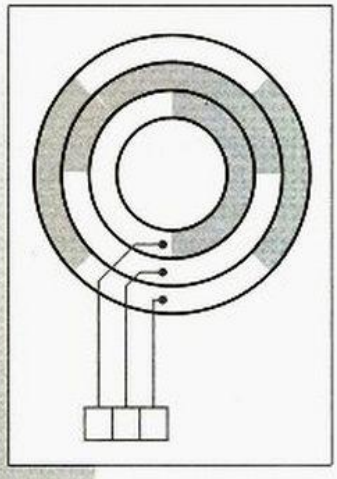
1.3 Defina los acrónimos siguientes: ASIC, CAD, CD, CO, CPLD, DIP, DVD, FPGA, HDL, CI, IP, LSI, MCM, MSI, NRE, OK, PBX, PCB, PLD, PWB, SMT, SSI, VHDL, VLSI.

1.4 Investigue las definiciones de los acrónimos que se enumeran a continuación: ABEL, CMOS, JPEG, MPEG, OK, PERL, VHDL. (¿En realidad OK es un acrónimo?)

1.5 Excluyendo los temas en la sección 1-2, enumere tres sistemas que alguna vez hayan sido analógicos y que se hayan convertido en digitales desde que usted está vivo.

1.6 Dibuje un circuito digital compuesto de una compuerta AND de 2 entradas y tres inversores, donde un inversor se encuentre conectado a cada una de las entradas de la compuerta AND y su salida. Para cada una de las cuatro posibles combinaciones de entradas que se aplican a las dos entradas primarias de este circuito, determine el valor producido para la salida primaria. ¿Existe un circuito más sencillo que suministre el mismo comportamiento de entrada/salida?

1.7 ¿Cuándo se deberían emplear los diagramas de terminales de la figura 1-5 en el diagrama esquemático de un circuito?



capítulo 2

Sistemas y códigos numéricos

Los sistemas digitales se construyen a partir de circuitos integrados que procesan dígitos binarios (ceros y unos), aunque muy pocos problemas de la vida real están basados en números binarios o cualquier tipo de números en absoluto. Como resultado, un diseñador de sistemas digitales debe establecer cierta correspondencia entre los dígitos binarios que procesan los circuitos digitales y los números, eventos y condiciones de la vida real. El propósito de este capítulo es demostrarle a usted cómo las cantidades numéricas familiares pueden representarse y manipularse en un sistema digital, y cómo los datos, eventos y condiciones no numéricos también pueden representarse.

Las primeras nueve secciones describen los sistemas numéricos binarios y muestran cómo se llevan a cabo las operaciones de suma, resta, multiplicación y división en estos sistemas. Las secciones 2.10 a 2.13 indican cómo se pueden codificar números decimales, caracteres de texto, posiciones mecánicas y condiciones arbitrarias, mediante el uso de cadenas de dígitos binarios.

La sección 2.14 presenta los "cubos *n*", los cuales proporcionan una forma de visualizar la relación entre diferentes cadenas de bits. Los cubos *n* son especialmente útiles en el estudio de los códigos de detección de error en la sección 2.15. Finalizamos el capítulo con una introducción a los códigos para transmisión y almacenamiento de datos de un bit a la vez.

2.1 Sistemas numéricos posicionales

sistema numérico posicional

El sistema numérico tradicional que aprendimos en la escuela y utilizamos todos los días en los negocios se conoce como un *sistema numérico posicional*. En un sistema de esta clase, un número se representa por medio de una cadena de dígitos, donde cada posición del dígito tiene un *peso* asociado. El valor de un número es una suma ponderada de los dígitos, por ejemplo:

peso o ponderación

$$1734 = 1 \cdot 1000 + 7 \cdot 100 + 3 \cdot 10 + 4 \cdot 1$$

Cada peso es una potencia de 10 que corresponde a la posición del dígito. Un punto decimal permite que se utilicen tanto potencias negativas como positivas de 10:

$$5185.68 = 5 \cdot 1000 + 1 \cdot 100 + 8 \cdot 10 + 5 \cdot 1 + 6 \cdot 0.1 + 8 \cdot 0.01$$

En general, un número D de la forma $d_1 d_0 . d_{-1} d_{-2}$ tiene el valor

$$D = d_1 \cdot 10^1 + d_0 \cdot 10^0 + d_{-1} \cdot 10^{-1} + d_{-2} \cdot 10^{-2}$$

base
raíz

Aquí, el 10 se denomina la *base* (*raíz, origen*) del sistema numérico. En un sistema numérico posicional general, la base puede ser cualquier entero $r \geq 2$, y un dígito en la posición i tiene un peso r^i. La forma de expresión de un número en un sistema de esta clase será

$$d_{p-1} d_{p-2} \cdots d_1 d_0 . d_{-1} d_{-2} \cdots d_{-n}$$

punto de la base

donde existen p dígitos a la izquierda del punto y n dígitos a la derecha de éste, llamado el *punto base*. Si este punto se omite, se supone que se encuentra a la derecha del dígito del extremo derecho. El valor del número es la suma de cada dígito multiplicado por el valor correspondiente de la base:

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

dígito de mayor orden
dígito más significativo
dígito de menor orden
dígito menos significativo

Excepto por la posibilidad de tener ceros al principio o al final, la representación de un número en un sistema numérico posicional es única. (Obviamente, 0185.6300 es igual a 185.63, y así por el estilo.) El dígito que está en el extremo izquierdo en un número de este tipo se denomina el *dígito de mayor orden* o *el dígito más significativo*; el que se encuentra en el extremo derecho es el *dígito de menor orden* o *el dígito menos significativo.*

dígito binario
bit
base binaria

Como aprendimos en el capítulo 3, los circuitos digitales tienen señales que se encuentran normalmente en una de dos condiciones: bajo o alto, cargado o descargado, encendido (on) o apagado (off). Las señales en estos circuitos se interpretan para representar *dígitos binarios* (o *bits*) que tienen uno de ambos valores, 0 y 1. De este modo, la *base binaria* se emplea normalmente para representar números en un sistema digital. La forma general de un número binario es

$$b_{p-1} b_{p-2} \cdots b_1 b_0 . b_{-1} b_{-2} \cdots b_{-n}$$

y su valor es

$$B = \sum_{i=-n}^{p-1} b_i \cdot 2^i$$

En un número binario, el punto base se denomina el *punto binario*. Cuando se trata con números binarios y otros números no decimales, utilizamos un subíndice para indicar la base de cada número, a menos que la base se sobreentienda en el contexto de trabajo. Ejemplos de números binarios y sus equivalentes decimales se ilustran a continuación.

punto binario

$$10011_2 = 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 19_{10}$$

$$100010_2 = 1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 34_{10}$$

$$101.001_2 = 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 + 0 \cdot 0.5 + 0 \cdot 0.25 + 1 \cdot 0.125 = 5.125_{10}$$

El bit que está en el extremo izquierdo de un número binario se conoce como el *bit de mayor orden* o el *bit más significativo* (*MSB*, por las siglas en inglés de *most significant bit*); el bit que está en el extremo derecho es el *bit de menor orden* o *bit menos significativo* (*LSB, least significant bit*).

MSB

LSB

2.2 Números octales y hexadecimales

La base 10 es importante debido a que la utilizamos en las operaciones cotidianas, mientras que la base 2 es importante porque los números binarios se pueden procesar directamente en los circuitos digitales. Los números en otras bases no se procesan directamente pero pueden ser importantes para documentación u otros propósitos. En particular, las bases 8 y 16 proporcionan representaciones convenientes abreviadas para números de múltiples bits en un sistema digital.

El *sistema numérico octal* utiliza la base 8, mientras que el *sistema numérico hexadecimal* emplea la base 16. La tabla 2-1 muestra los enteros binarios desde 0 hasta 1111 y sus equivalentes octal, decimal y hexadecimal. El sistema octal necesita 8 dígitos, de manera que utiliza los dígitos del 0 al 7 del sistema decimal. El sistema hexadecimal necesita 16 dígitos, de modo que combina los dígitos decimales del 0 al 9 con las letras *A* hasta la *F*.

sistema numérico octal

sistema numérico hexadecimal

dígitos hexadecimales A–F

Los sistemas numéricos octal y hexadecimal son útiles para representar números de bits múltiples, debido a que sus bases son potencias de 2. Puesto que una cadena de tres bits puede tomarse en ocho diferentes combinaciones, se sigue que cada cadena de 3 bits puede representarse de manera única mediante un dígito octal, de acuerdo con la tercera y la cuarta columnas de la tabla 2-1. De manera semejante, una cadena de 4 bits puede ser representada por un dígito hexadecimal de acuerdo con las columnas quinta y sexta de la tabla.

De este modo, es muy fácil convertir un número binario en octal. Comenzando desde el punto binario y viajando hacia la izquierda, simplemente separamos los bits en grupos de tres y reemplazamos cada grupo con el correspondiente dígito octal:

conversión de binario a octal

$$1000110011110_2 = 100\ 011\ 001\ 110_2 = 4316_8$$

$$1110110111010100 1_2 = 011\ 101\ 101\ 110\ 101\ 001_2 = 355651_8$$

El procedimiento para la conversión de binario a hexadecimal es semejante, excepto que utilizamos grupos de cuatro bits:

conversión de binario a hexadecimal

$$1000110011110_2 = 1000\ 1100\ 1110_2 = 8CE_{16}$$

$$1110110111010100 1_2 = 0001\ 1101\ 1011\ 1010\ 1001_2 = 1DBA9_{16}$$

En estos ejemplos, hemos agregado libremente ceros a la izquierda para hacer el número total de bits un múltiplo de 3 o 4 como se requiere.

Tabla 2-1
Números binarios, decimales, octales y hexadecimales.

Binario	*Decimal*	*Octal*	*Cadena de 3 bits*	*Hexadecimal*	*Cadena de 4 bits*
0	0	0	000	0	0000
1	1	1	001	1	0001
10	2	2	010	2	0010
11	3	3	011	3	0011
100	4	4	100	4	0100
101	5	5	101	5	0101
110	6	6	110	6	0110
111	7	7	111	7	0111
1000	8	10	—	8	1000
1001	9	11	—	9	1001
1010	10	12	—	A	1010
1011	11	13	—	B	1011
1100	12	14	—	C	1100
1101	13	15	—	D	1101
1110	14	16	—	E	1110
1111	15	17	—	F	1111

Si un número binario contiene dígitos a la derecha del punto binario, podemos convertirlo a octal o hexadecimal al comenzar por el punto binario y movernos hacia la derecha. Tanto el lado izquierdo como el derecho pueden rellenarse con ceros para obtener múltiplos de tres o cuatro bits, como se muestra en el ejemplo que se presenta a continuación:

$$10.1011001011_2 = 010\,.\,101\ 100\ 101\ 100_2 = 2.5454_8$$
$$= 0010\,.\,1011\ 0010\ 1100_2 = 2.B2C_{16}$$

conversión octal o hexadecimal a binaria

Realizar la conversión en la dirección inversa, partiendo de octal o hexadecimal hacia binario, es algo muy sencillo. Simplemente reemplazamos cada dígito octal o hexadecimal con la correspondiente cadena de 3 o 4 bits, como se muestra a continuación:

$$1357_8 = 001\ 011\ 101\ 111_2$$
$$2046.17_8 = 010\ 000\ 100\ 110\,.\,001\ 111_2$$
$$BEAD_{16} = 1011\ 1110\ 1010\ 1101_2$$
$$9F.46C_{16} = 1001\ 111\,.\,0100\ 0110\ 1100_2$$

El sistema numérico octal fue bastante popular hace 15 años debido a que ciertas minicomputadoras tenían sus luces e interruptores frontales acomodados en grupos de tres. Sin embargo, el sistema numérico octal no se utiliza mucho en la actualidad, a consecuencia de la preponderancia de las máquinas que procesan *bytes* compuestos de 8 bits. Es difícil extraer los valores de byte individual en cantidades de bytes múltiples en

byte (octeto)

CUANDO TENGA 64 AÑOS

A medida que usted envejece descubrirá que el sistema numérico hexadecimal es útil no sólo para las computadoras. Cuando llegué a los 40, le dije a mis amigos que acababa de cumplir 28_{16}. Pronuncié el "$_{16}$" en forma que casi no se escuchara, naturalmente. A los 50, tendré solamente 32_{16}.

Las personas se emocionan en torno a las décadas cuando se trata de celebrar cumpleaños, como 20, 30, 40, 50..., pero usted podría convencer a sus amistades de que el sistema decimal no es de importancia fundamental. Los cambios más significativos de la existencia ocurren alrededor de los cumpleaños números 2, 4, 8, 16, 32 y 64, que es cuando usted agrega un bit más significativo a su edad. O, ¿por qué piensa usted que los Beatles cantaban "Cuando cumpla sesenta y *cuatro*"?

la representación octal; por ejemplo, ¿cuáles son los valores en octal de los cuatro bytes de 8 bits en el número de 32 bits cuya representación octal es 12345670123_8?

En el sistema hexadecimal, dos dígitos representan un byte de 8 bits y $2n$ dígitos representan una palabra de n bytes; cada par de dígitos constituyen exactamente un byte. Por ejemplo, el número hexadecimal de 32 bits $5678ABCD_{16}$ se compone de cuatro bytes con valores 56_{16}, 78_{16}, AB_{16} y CD_{16}. En este contexto, un dígito hexadecimal de 4 bits se denomina a veces un *nibble (medio byte)*; un número de 32 bits (4 bytes) tiene ocho nibbles. Los números hexadecimales se utilizan con cierta frecuencia para describir un espacio de dirección de la memoria de la computadora. Por ejemplo, una computadora con direcciones de 16 bits puede describirse como que tiene memoria de lectura/escritura instalada en las direcciones $0\text{-}EFFF_{16}$ y memoria sólo de lectura en las direcciones $F000\text{-}FFFF_{16}$. Muchos lenguajes de programación para computadoras utilizan el prefijo "0x" para denotar un número hexadecimal, por ejemplo, `0xBFC0000`.

nibble (medio byte o medio octeto)

prefijo `0x`

2.3 Conversiones generales de sistema numérico posicional

En general, la conversión entre dos bases no puede hacerse por simple sustitución; se requieren operaciones aritméticas. En esta sección mostraremos cómo convertir un número en cualquier base a la base 10 y viceversa, haciendo uso de aritmética de base 10.

En la sección 2.1, indicamos que el valor de un número en cualquier base está dado por la fórmula

conversión de base-r a decimal

$$D = \sum_{i=-n}^{p-1} d_i \cdot r^i$$

donde r es la base del número y existen p dígitos a la izquierda del punto base y n a la derecha. De esta forma, el valor del número puede encontrarse al convertir cada dígito del número a su equivalente en base 10, y expandir la fórmula utilizando aritmética de base 10. A continuación se proporcionan algunos ejemplos:

$$1CE8_{16} = 1\cdot16^3 + 12\cdot16^2 + 14\cdot16^1 + 8\cdot16^0 = 7400_{10}$$
$$F1A3_{16} = 15\cdot16^3 + 1\cdot16^2 + 10\cdot16^1 + 3\cdot16^0 = 61859_{10}$$
$$436.5_8 = 4\cdot8^2 + 3\cdot8^1 + 6\cdot8^0 + 5\cdot8^{-1} = 286.625_{10}$$
$$132.3_4 = 1\cdot4^2 + 3\cdot4^1 + 2\cdot4^0 + 3\cdot4^{-1} = 30.75_{10}$$

fórmula de expansión anidada

Un atajo para convertir números enteros a base 10 puede obtenerse al volver a escribir la fórmula de expansión de manera anidada:

$$D = ((\cdots((d_{p-1})\cdot r + d_{p-2})\cdot r + \cdots)\cdots r + d_1)\cdot r + d_0$$

Esto es, comenzamos con una suma de 0; iniciando con el dígito que está en el extremo izquierdo, multiplicamos la suma por r, y agregamos el siguiente dígito a la suma, repetimos este proceso hasta que todos los dígitos hayan sido procesados. Por ejemplo, podemos escribir

$$\mathrm{F1AC}_{16} = (((15)\cdot 16 + 1)\cdot 16 + 10)\cdot 16 + 12$$

conversión de decimal a base r

Esta fórmula se emplea en algoritmos de conversión programados e iterativos (como los de la tabla 4-38 en la página 279). También es el fundamento de un método muy conveniente para convertir un número decimal D a una base r. Considere lo que ocurre si dividimos la fórmula entre r. Puesto que la parte entre paréntesis de la fórmula es igualmente divisible entre r, el cociente será

$$Q = (\cdots((d_{p-1})\cdot r + d_{p-2})\cdot r + \cdots)\cdot r + d_1$$

y el residuo será d_0. De este modo, d_0 puede calcularse como el residuo de la división larga de D entre r. Adicionalmente, el cociente Q tiene la misma forma que la fórmula original. Por lo tanto, divisiones sucesivas entre r nos proporcionan dígitos sucesivos de D de derecha a izquierda, hasta que todos los dígitos de D hayan sido derivados. Acto seguido, se muestran ejemplos que ilustran lo anterior:

$179 \div 2 = 89$ residuo 1 (LSB)
$\div 2 = 44$ residuo 1
$\div 2 = 22$ residuo 0
$\div 2 = 11$ residuo 0
$\div 2 = 5$ residuo 1
$\div 2 = 2$ residuo 1
$\div 2 = 1$ residuo 0
$\div 2 = 0$ residuo 1 (MSB)

$179_{10} = 10110011_2$

$467 \div 8 = 58$ residuo 3 (dígito significativo menor)
$\div 8 = 7$ residuo 2
$\div 8 = 0$ residuo 7 (dígito significativo mayor)

$467_{10} = 723_8$

$3417 \div 16 = 213$ residuo 9 (dígito significativo menor)
$\div 16 = 13$ residuo 5
$\div 16 = 0$ residuo 13 (dígito significativo mayor)

$3417_{10} = \mathrm{D59}_{16}$

La tabla 2-2 resume los métodos para la conversión entre las bases más comunes.

Tabla 2-2 Métodos de conversión para bases comunes.

Conversión	*Método*	*Ejemplo*
Binario a		
Octal	Sustitución	$10111011001_2 = 10\ 111\ 011\ 001_2 = 2731_8$
Hexadecimal	Sustitución	$10111011001_2 = 101\ 1101\ 1001_2 = 5D9_{16}$
Decimal	Suma	$10111011001_2 = 1 \cdot 1024 + 0 \cdot 512 + 1 \cdot 256 + 1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 1497_{10}$
Octal a		
Binario	Sustitución	$1234_8 = 001\ 010\ 011\ 100_2$
Hexadecimal	Sustitución	$1234_8 = 001\ 010\ 011\ 100_2 = 0010\ 1001\ 1100_2 = 29C_{16}$
Decimal	Suma	$1234_8 = 1 \cdot 512 + 2 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 668_{10}$
Hexadecimal a		
Binario	Sustitución	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2$
Octal	Sustitución	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2 = 1\ 100\ 000\ 011\ 011\ 110_2 = 140336_8$
Decimal	Suma	$C0DE_{16} = 12 \cdot 4096 + 0 \cdot 256 + 13 \cdot 16 + 14 \cdot 1 = 49374_{10}$
Decimal a		
Binario	División	$108_{10} \div 2 = 54$ residuo 0 (LSB) $\div 2 = 27$ residuo 0 $\div 2 = 13$ residuo 1 $\div 2 = 6$ residuo 1 $\div 2 = 3$ residuo 0 $\div 2 = 1$ residuo 1 $\div 2 = 0$ residuo 1 (MSB) $108_{10} = 1101100_2$
Octal	División	$108_{10} \div 8 = 13$ residuo 4 (dígito menos significativo) $\div 8 = 1$ residuo 5 $\div 8 = 0$ residuo 1 (dígito más significativo) $108_{10} = 154_8$
Hexadecimal	División	$108_{10} \div 16 = 6$ residuo 12 (dígito menos significativo) $\div 16 = 0$ residuo 6 (dígito más significativo) $108_{10} = 6C_{16}$

Tabla 2-3
Suma binaria y tabla de restas.

c_{ent} o b_{ent}	x	y	c_{sal}	s	b_{sal}	d
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	1
0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

2.4 Suma y resta de números no decimales

La suma y la resta de números no decimales con el método manual utiliza la misma técnica que aprendimos en la escuela primaria para los números decimales; la única trampa es que las tablas de suma y resta son diferentes.

suma binaria

La tabla 2-3 es la tabla de suma y resta para dígitos binarios. Para sumar dos números binarios X y Y, sumamos juntos los bits menos significativos con un acarreo inicial (c_{ent}) de 0, produciendo bits de acarreo (c_{sal}) y de suma (s) de acuerdo con la tabla. Continuamos procesando bits de derecha a izquierda, sumando el acarreo fuera de cada columna a la suma de la siguiente columna.

Dos ejemplos de sumas decimales y las correspondientes sumas binarias se muestran en la figura 2-1, los dígitos en negritas indican el acarreo de 1. Los mismos ejemplos se repiten a continuación junto con dos más, con los acarreos mostrados como una cadena de bits C:

C		**101111000**	C		**001011000**
X	190	10111110	X	173	**10101101**
Y	+141	+ 10001101	Y	+ 44	+ **00101100**
$X+Y$	331	101001011	$X+Y$	217	11011001
C		**011111110**	C		**000000000**
X	127	01111111	X	170	**10101010**
Y	+ 63	+ 00111111	Y	+ 85	+ **01010101**
$X+Y$	190	10111110	$X+Y$	255	11111111

resta binaria

minuendo
sustraendo

La resta binaria se realiza de modo similar, empleando "préstamos" (b_{ent} y b_{sal}) en lugar de acarreos entre pasos, y produciendo un bit de diferencia d. Dos ejemplos de restas decimales y las correspondientes restas binarias se muestran en la figura 2-2. Como en la resta decimal, los valores del minuendo binario en las columnas se modifican cuando se presenta el préstamo, como se ilustra mediante las flechas y los bits que aparecen

			1	1	1	1						1		1	1				
X	190		1	0	1	1	1	1	1	0	X	173		1	0	1	0	1	1	0	1
Y	+141	+	1	0	0	0	1	1	0	1	Y	+ 44	+	0	0	1	0	1	1	0	0
$X+Y$	331		1	0	1	0	0	1	0	1	1	$X+Y$	217		1	1	0	1	1	0	0	1

Figura 2-1 Ejemplos de sumas decimales y sus correspondientes.

en negritas. Los ejemplos de la figura se repiten a continuación con dos más, mostrando esta vez los préstamos como una cadena de bits B:

B		**001111100**	***B***		**011011010**
X	229	11100101	X	210	11010010
Y	− 46	− 00101110	Y	−109	− 01101101
$X-Y$	183	10110111	$X-Y$	101	01100101
B		**010101010**	***B***		**000000000**
X	170	10101010	X	221	11011101
Y	− 85	− 01010101	Y	− 76	− 01001100
$X-Y$	85	01010101	$X-Y$	145	10010001

comparación de números

Un uso muy común de la resta en computadoras es la comparación de dos números. Por ejemplo, si la operación $X - Y$ produce un préstamo (o transporte negativo) que procede de la posición del bit más significativo, entonces X es menor que Y; de otro modo, X es mayor o igual que Y. La relación entre acarreos y préstamos en sumadores y restadores se explorará en la sección 5.10.

Las tablas de suma y resta pueden desarrollarse para dígitos octales y hexadecimales, o cualquier otra base deseada. Sin embargo, pocos ingenieros en computación se molestan en memorizar estas tablas. Si usted necesita manipular números no decimales, entonces será fácil en esas ocasiones convertirlos a formato decimal, calcular resultados,

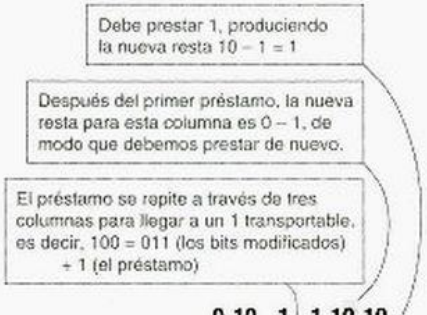


Figura 2-2 Ejemplos de restas decimales y sus correspondientes restas binarias.

							0	10	1		1	10	10																					
minuendo	X	229			1	1	1	0	0	1	0	1		X	210			1	1	0	1	0	0	1	0				0	10	10	0	1	10	0 10
sustraendo	Y	− 46		−	0	0	1	0	1	1	1	0		Y	−109		−	0	1	1	0	1	1	0	1				1	1	0	1	0	0	1	0
diferencia	$X-Y$	183			1	0	1	1	0	1	1	1		$X-Y$	101			0	1	1	0	0	1	0	1			−	0	1	1	0	1	1	0 1

y convertirlos de vuelta otra vez. Por otro lado, si debe realizar cálculos en formato binario, octal o hexadecimal con bastante frecuencia, entonces debería pedirle a Santa Claus una calculadora hexadecimal para programadores de Texas Instruments o de Casio.

Si las baterías de la calculadora se agotan, puede utilizar algunos atajos mentales para facilitar la aritmética no decimal. En general, cada suma (o resta) de columna puede hacerse al convertir los dígitos de la columna a formato decimal, sumar en decimal, y posteriormente convertir el resultado a la suma correspondiente y dígitos de acarreo en la base no decimal (un acarreo se produce siempre que la suma de la columna es igual o mayor a la base). Puesto que la suma se hace en formato decimal, confiamos en nuestros conocimientos de la tabla decimal de adición; la única cosa novedosa que necesitamos aprender es la conversión de dígitos decimales a no decimales y viceversa. La secuencia de pasos para sumar mentalmente dos números hexadecimales se muestra a continuación:

suma hexadecimal

C		**1**	**1**	**0**	**0**		**1**	**1**	**0**	**0**
X		1	9	B	9_{16}		1	9	11	9
Y	+	C	7	E	6_{16}	+	12	7	14	6
X+Y		E	1	9	F_{16}		14	17	25	15
							14	16+1	16+9	15
							E	1	9	F

2.5 Representación de números negativos

Hasta ahora, solamente hemos tratado con números positivos, pero existen muchas formas de representar números negativos. En los negocios utilizamos el sistema de magnitud con signo, que se discute más adelante. Sin embargo, la mayor parte de las computadoras emplean alguno de los sistemas numéricos de complemento que presentaremos posteriormente.

2.5.1 Representación de magnitud con signo

sistema de magnitud con signo

En el *sistema de magnitud con signo*, un número se compone de una magnitud y de un símbolo que indica si la magnitud es positiva o negativa. De esta forma, interpretamos los números decimales +98, –57, +123.5 y –13 de la manera habitual, y también suponemos que el signo es "+" si no aparece ningún símbolo escrito. Existen dos posibles representaciones de cero, "+0" y "–0", pero ambas tienen el mismo valor.

bit de signo

El sistema de magnitud con signo se aplica a los números binarios haciendo uso de una posición de bit extra para representar el signo (el *bit de signo*). Tradicionalmente, el bit más significativo (MSB, por sus siglas en inglés) de una cadena de bits es empleado como el bit de signo (0 = signo más, 1 = signo menos), y los bits de menor orden contienen la magnitud. Así, podemos escribir varios enteros de 8 bits con magnitud con signo y sus equivalentes decimales:

$$01010101_2 = +85_{10} \qquad 11010101_2 = -85_{10}$$
$$01111111_2 = +127_{10} \qquad 11111111_2 = -127_{10}$$
$$00000000_2 = +0_{10} \qquad 10000000_2 = -0_{10}$$

El sistema de magnitud con signo tiene un número idéntico de enteros positivos y negativos. Un entero de magnitud con signo de n bits está situado dentro del intervalo que va desde $-(2^{n-1}-1)$ hasta $+(2^{n-1}-1)$ y existen dos representaciones posibles del cero.

sumador de magnitud con signo

Ahora supongamos que deseamos construir un circuito lógico digital que sume números de magnitud con signo. El circuito debe examinar los signos de los sumandos para determinar qué hacer con las magnitudes. Si los signos son los mismos, debe sumar las magnitudes y proporcionar el resultado con el mismo signo. Si los signos son diferentes, debe comparar las magnitudes, restar el más pequeño del mayor y proporcionar al resultado el signo del más grande. Todos estos "si" condicionales, "sumas", "restas" y "comparaciones" se traducen en una gran cantidad de circuitos lógicos complejos. Los sumadores para los sistemas numéricos de complemento son mucho más simples, como demostraremos a continuación. Quizás la única característica que redime a un sistema de magnitud con signo es que, una vez que sabemos cómo construir un sumador de magnitud con signo, la fabricación de un restador de magnitud con signo es algo casi trivial: solamente se debe cambiar el signo del sustraendo y pasarlo junto con el minuendo a un sumador.

restador de magnitud con signo

2.5.2 Sistemas numéricos de complemento

sistema numérico de complemento

Mientras que el sistema de magnitud con signo convierte en negativo un número al cambiar su signo, un *sistema numérico de complemento* convierte en negativo un número tomando su complemento como definido por el sistema. Tomar el complemento es más difícil que cambiar el signo, pero dos números en un sistema numérico de complemento pueden sumarse o restarse directamente sin tener que realizar las verificaciones de magnitud y signo que requiere el sistema de magnitud con signo. Describiremos dos sistemas numéricos de complemento, llamados el "complemento de base" y el "complemento de base reducida".

En cualquier sistema numérico de complemento, normalmente tratamos con un número fijo de dígitos, digamos n. (Sin embargo, podemos aumentar el número de dígitos mediante "extensión de signo" como se muestra en el ejercicio 2.23, y disminuir el número mediante el truncamiento de los dígitos de orden mayor como se muestra en el ejercicio 2.24.) Suponemos adicionalmente que la base es r, y que los números tienen la forma

$$D = d_{n-1}d_{n-2}\cdots d_1 d_0.$$

El punto de base se encuentra a la derecha y por tanto el número es un entero. Si una operación produce un resultado que requiera más de n dígitos, eliminamos el (los) dígito(s) extra de mayor orden. Si un número D se complementa dos veces, el resultado será D.

2.5.3 Representación de complemento de base

sistema de complemento de base

En un *sistema de complemento de base*, el complemento de un número de n dígitos se obtiene al restarlo de r^n. En el sistema numérico decimal, el complemento de base se denomina *complemento de 10*. Algunos ejemplos utilizando números decimales de 4 dígitos (y resta de 10,000) se muestran en la tabla 2–4.

complemento a 10

Por definición, el complemento de base de un número D de n dígitos se obtiene al restarlo de r^n. Si D se encuentra entre 1 y $r^n - 1$, esta resta produce otro número entre 1 y

Tabla 2-4
Ejemplos de complementos de 10 y 9.

Número	*Complemento de 10*	*Complemento de 9*
1849	8151	8150
2067	7933	7932
100	9900	9899
7	9993	9992
8151	1849	1848
0	10000 (= 0)	9999

$r^n - 1$. Si D es 0, el resultado de la resta es r^n, lo cual tiene la forma $100 \cdots 00$, donde hay un total de $n + 1$ dígitos. Descartamos el dígito extra de mayor orden y obtenemos el resultado 0. Por consiguiente, sólo existe una representación de cero en un sistema de complemento de base.

cálculo del complemento de base

Parece de la definición que una operación de resta es necesaria para calcular el complemento de base de D. Sin embargo, esta resta puede evitarse al volver a escribir r^n como $(r^n - 1) + 1$ y $r^n - D$ como $((r^n - 1) - D) + 1$. El número $r^n - 1$ tiene la forma $mm \cdots mm$, donde $m = r - 1$ y hay n cantidad de m's. Por ejemplo, 10,000 es igual a 9,999 + 1. Si definimos el complemento de un dígito d como $r - 1 - d$, entonces $(r^n - 1) - D$ se obtiene mediante la complementación de los dígitos de D. Por consiguiente, el complemento de base de un número D se obtiene al complementar los dígitos individuales de

Tabla 2-5
Complementos de dígito.

	Complemento			
Dígito	*Binario*	*Octal*	*Decimal*	*Hexadecimal*
0	1	7	9	F
1	0	6	8	E
2	–	5	7	D
3	–	4	6	C
4	–	3	5	B
5	–	2	4	A
6	–	1	3	9
7	–	0	2	8
8	–	–	1	7
9	–	–	0	6
A	–	–	–	5
B	–	–	–	4
C	–	–	–	3
D	–	–	–	2
E	–	–	–	1
F	–	–	–	0

D y sumar 1. Por ejemplo, el complemento de 10 de 1849 es 8150 + 1, o 1851. Debería confirmarse que este truco también funciona para los ejemplos anteriores de complementos de 10. La tabla 2-5 enumera los complementos de dígito para los números binarios, octales, decimales y hexadecimales.

2.5.4 Representación de complemento a dos

complemento a dos

Para números binarios, el complemento de base se conoce como el *complemento a dos*. El MSB de un número en este sistema sirve como el bit de signo; un número es negativo si y sólo si su MSB es 1. El equivalente decimal para un número binario de complemento a dos se calcula de la misma forma que para un número sin signo, excepto que el peso del MSB es -2^{n-1} en vez de $+2^{n-1}$. El intervalo de los números representables abarca desde $-(2^{n-1})$ hasta $+(2^{n-1} - 1)$. A continuación se muestran algunos ejemplos de 8 bits:

peso (ponderación) del MSB

$17_{10} = 00010001_2$
$\Downarrow$ **bits de complemento**
11101110
$+1$
$11101111_2 = -17_{10}$

$-99_{10} = 10011101_2$
$\Downarrow$ **bits de complemento**
01100010
$+1$
$01100011_2 = 99_{10}$

$119_{10} = 01110111$
$\Downarrow$ **bits de complemento**
10001000
$+1$
$10001001_2 = -119_{10}$

$-127_{10} = 10000001$
$\Downarrow$ **bits de complemento**
01111110
$+1$
$01111111_2 = 127_{10}$

$0_{10} = 00000000_2$
$\Downarrow$ **bits de complemento**
11111111
$+1$
$\mathbf{1}\ 00000000_2 = 0_{10}$

$-128_{10} = 10000000_2$
$\Downarrow$ **bits de complemento**
01111111
$+1$
$10000000_2 = -128_{10}$

Un acarreo fuera de la posición del MSB ocurre en un caso, como se muestra en negritas. Como en todas las operaciones de complemento a dos, este bit se ignora y solamente se utilizan los n bits de menor orden del resultado.

En el sistema numérico de complemento a dos, el cero se considera positivo puesto que su bit de signo es 0. Puesto que el complemento a dos tiene solamente una representación del cero, acabamos con un número negativo extra, $-(2^n - 1)$ que no tiene un homólogo positivo.

número negativo extra

Podemos convertir un número X, de n bits de complemento a dos en uno de m bits, pero se necesita algo de cuidado. Si $m > n$, debemos agregar $m - n$ copias del bit de signo de X a la izquierda de X (*véase* el ejercicio 2.23). Es decir, rellenamos un número positivo con ceros y un uno negativo con unos; esto se conoce como *extensión de signo*. Si $m < n$, descartamos los $m - n$ bits que están a la extrema izquierda de X; sin embargo, el

extensión de signo

resultado es válido solamente si todos los bits descartados son iguales que el bit de signo del resultado (véase el ejercicio 2.24).

La mayoría de las computadoras y otros sistemas digitales utilizan el sistema de complemento a dos para representar números negativos. No obstante, para completar nuestro estudio, también describiremos los sistemas de complemento de base reducida y de complemento a unos.

*2.5.5 Representación de complemento de base reducida

sistema de complemento de base reducida

complemento de 9s

En un *sistema de complemento de base reducida*, el complemento de un número de n dígitos D se obtiene al restarlo de $r^n - 1$. Esto puede conseguirse al complementar los dígitos individuales de D, *sin* sumar 1, como sucede en el sistema de complemento de base. En representación decimal, esto se denomina *complemento de 9s*; se proporcionan algunos ejemplos en la última columna de la tabla 2-4 de la página 36.

*2.5.6 Representación de complemento a unos

complemento a unos

El sistema de complemento de base reducida para números binarios se conoce como el *complemento a unos*. Como en el complemento a dos, el bit más significativo es el signo, 0 si es positivo y 1 si es negativo. De este modo hay dos representaciones del cero, cero positivo ($00\cdots00$) y cero negativo ($11\cdots11$). Las representaciones de los números positivos son las mismas tanto para el complemento a dos como para el complemento a unos. Sin embargo, las representaciones de los números negativos difieren por 1. Un peso de $-(2^{n-1}-1)$, en vez de -2^{n-1}, se otorga al bit más significativo cuando se calcula el equivalente decimal de un número de complemento a unos. El intervalo de números representables va de $-(2^{n-1}-1)$ hasta $+(2^{n-1}-1)$. Algunos números de 8 bits y sus complementos de uno se muestran enseguida:

$$17_{10} = 00010001_2 \qquad -99_{10} = 10011100_2$$
$$\Downarrow \qquad\qquad \Downarrow$$
$$11101110_2 = -17_{10} \qquad 01100011_2 = 99_{10}$$
$$119_{10} = 01110111_2 \qquad -127_{10} = 10000000_2$$
$$\Downarrow \qquad\qquad \Downarrow$$
$$10001000_2 = -119_{10} \qquad 01111111_2 = 127_{10}$$
$$0_{10} = 00000000_2 \text{ (cero positivo)}$$
$$\Downarrow$$
$$11111111_2 = 0_{10} \text{ (cero negativo)}$$

Las principales ventajas del sistema de complemento a unos es su simetría y la facilidad de complementación. Sin embargo, el diseño del sumador para números de complemento a unos es algo más delicado que un sumador de complemento a dos (véase el ejercicio 7.72). Además, los circuitos de detección de cero en un sistema de com-

* En todo el libro, las secciones opcionales están marcadas con un asterisco.

plemento a unos deben verificar ambas representaciones de cero, o convertir siempre $11\cdots11$ a $00\cdots00$.

*2.5.7 Representaciones por exceso

Sí, el número de sistemas diferentes para la representación de números negativos *es* enorme, pero existe una más que debemos incluir. En la *representación de exceso B*, una cadena de m bits cuyo valor de entero sin signo es $M(0 \le M < 2^m)$representa el entero con signo $M - B$, donde B se denomina la *tendencia* del sistema numérico.

representación de exceso B

tendencia

Por ejemplo, un *sistema de exceso* 2^{m-1} representa cualquier número X en el intervalo de -2^{m-1} a $+2^{m-1} - 1$ mediante la representación binaria de m bits de $X + 2^{m-1}$(que siempre es no negativa y menor que 2^m). El intervalo de esta representación es exactamente la misma que en el caso de los números de complemento a dos de m bits. De hecho, las representaciones de cualquier número en ambos sistemas son idénticas, excepto por los bits de signo, que siempre son opuestos. (Advierta que esto es verdad solamente cuando la tendencia es 2^{m-1}.)

sistema de exceso de 2^{m-1}

El uso más común de las representaciones de exceso se encuentra en los sistemas numéricos de punto flotante (véase la parte de Referencias).

2.6 Suma y resta de complemento a dos

2.6.1 Reglas de la suma

Una tabla de números decimales y sus equivalentes en diferentes sistemas numéricos, tabla 2-6, revela por qué se prefiere el complemento a dos en las operaciones aritméticas. Si comenzamos con 1000_2 (-8_{10}) y contamos de manera ascendente, vemos que cada número sucesivo de complemento a dos en todo el recorrido hasta 0111_2 $(+7_{10})$ puede obtenerse mediante la suma de 1 al anterior, ignorando cualquier acarreo más allá de la posición del cuarto bit. No puede decirse lo mismo de los números de complemento a unos y de magnitud con signo. Debido a que la suma ordinaria es simplemente una extensión del conteo, los números de complemento a dos pueden sumarse mediante la suma binaria ordinaria, ignorando cualquier acarreo más allá del MSB. El resultado siempre será la suma correcta siempre y cuando no se exceda el intervalo del sistema numérico. Algunos ejemplos de suma decimal y las correspondientes sumas de complemento a dos de 4 bits confirman esto:

suma de complemento a dos

+3	0011	−2	1110
+ +4	+ 0100	+ −6	+ 1010
+7	0111	−8	11000
+6	0110	+4	0100
+ −3	+ 1101	+ −7	+ 1001
+3	10011	−3	1101

Tabla 2-6 Números decimales y de 4 bits.

Decimal	*Complemento a dos*	*Complemento a unos*	*Magnitud con signo*	*Exceso de* 2^{m-1}
−8	1000	—	—	0000
−7	1001	1000	1111	0001
−6	1010	1001	1110	0010
−5	1011	1010	1101	0011
−4	1100	1011	1100	0100
−3	1101	1100	1011	0101
−2	1110	1101	1010	0110
−1	1111	1110	1001	0111
0	0000	1111 o 0000	1000 o 0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

2.6.2 Una visión gráfica

Otra manera de ver el sistema de complemento a dos hace uso del "contador" de 4 bits que se ilustra en la figura 2-3. Aquí hemos ilustrado los números en una representación circular o "modular". La operación de este contador imita muy de cerca la de un circuito contador en ambos sentidos en la vida real, el cual estudiaremos en la sección 8.4. Si empezamos con la flecha que apunta a cualquier número, podemos sumar $+n$ a ese número al contar de manera ascendente n veces, es decir, moviendo la flecha n posiciones

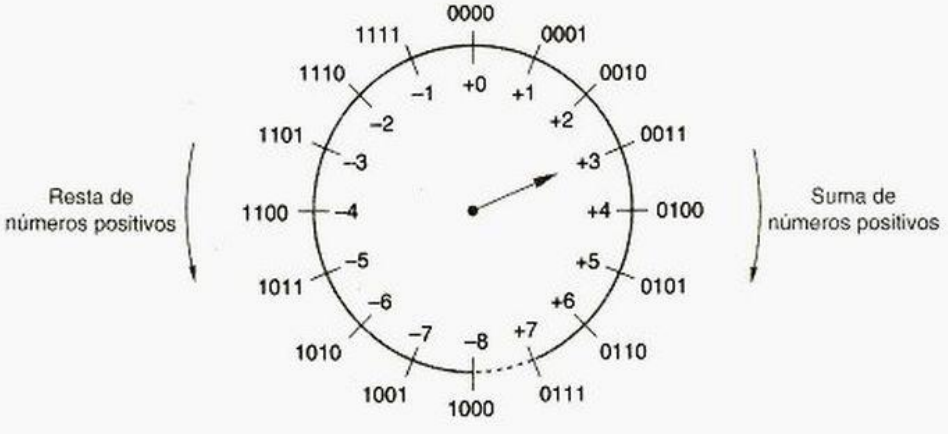


Figura 2-3
Una representación de conteo modular de números de complemento a dos de 4 bits.

en el sentido de giro de las manecillas del reloj. También es evidente que podemos restar n de un número al contar de forma descendente n veces, es decir, al mover la flecha n posiciones en sentido contrario de las manecillas del reloj. Naturalmente, estas operaciones proporcionan resultados correctos solamente si n es lo suficientemente pequeño de modo que no cruce la discontinuidad que existe entre –8 y +7.

Lo que es más interesante es que también podemos restar n (o sumar $-n$) al mover la flecha $16 - n$ posiciones en el sentido de giro de las manecillas del reloj. Nótese que la cantidad $16 - n$ es lo que definimos como el complemento a dos de 4 bits de n, es decir, la representación de complemento a dos de $-n$. Esto apoya gráficamente nuestra afirmación anterior, en la que un número negativo en la representación de complemento a dos puede ser sumado a otro número simplemente sumando las representaciones de 4 bits utilizando la suma binaria ordinaria. Sumar un número en la figura 2–3 es equivalente a mover la flecha un número correspondiente de posiciones en sentido contrario al de las manecillas del reloj.

2.6.3 Desbordamiento

Si una operación de suma produce un resultado que excede el intervalo del sistema numérico, se dice que ocurre un *desbordamiento*. En la representación de conteo modular de la figura 2-3, el desbordamiento se presenta durante la suma de números positivos cuando contamos pasando +7. La suma de dos números con signos diferentes nunca puede producir desbordamiento, pero la suma de dos números de signo semejante puede hacerlo, como se muestra en los ejemplos siguientes:

desbordamiento

–3	1101		+5	0101	
+ –6	+ 1010		+ +6	+ 0110	
–9	10111	= +7	+11	1011	= –5
–8	1000		+7	0111	
+ –8	+ 1000		+ +7	+ 0111	
–16	10000	= +0	+14	1110	= –2

Afortunadamente, existe una regla simple para detectar el desbordamiento en la suma: una suma provoca desbordamiento si los signos de los sumandos son los mismos y el signo de la suma es diferente del signo de los sumandos. La regla del desbordamiento se establece en ocasiones en términos de acarreos que se generan durante la operación de suma: una suma provoca desbordamiento si los bits de acarreo de entrada c_{ent} y de salida c_{sal} de la posición del signo son diferentes. Un examen detallado de la tabla 2-3 en la página 32 muestra que ambas reglas son equivalentes: solamente existen dos casos donde $c_{ent} \neq c_{sal}$ y éstos son los dos únicos casos donde $x = y$, y el bit de suma es diferente.

reglas del desbordamiento

2.6.4 Reglas de la resta

Los números de complemento a dos pueden restarse como si fueran números binarios ordinarios sin signo, y pueden formularse reglas apropiadas para detectar un desbordamiento. Sin embargo, la mayor parte de los circuitos de resta para números de complemento a dos no realizan la resta en forma directa. En vez de ello, hacen negativo el sustraendo tomando su complemento a dos y posteriormente lo suman al minuendo aplicando las reglas normales para la suma.

resta de complemento a dos

La negación del sustraendo y su adición al minuendo pueden llevarse a cabo con solamente una operación de suma como se explica a continuación: realice un complemento bit a bit del sustraendo y sume el sustraendo complementado al minuendo con un acarreo de entrada (c_{ent}) de 1 en lugar de 0. A continuación presentamos algunos ejemplos:

		$1 - c_{ent}$			$1 - c_{ent}$
+4	0100	0100	+3	0011	0011
− +3	− 0011	+ 1100	− +4	− 0100	+ 1011
+1		10001	−1		1111

		$1 - c_{ent}$			$1 - c_{ent}$
+3	0011	0011	−3	1101	1101
− −4	− 1100	+ 0011	− −4	− 1100	+ 0011
+7		0111	+1		10001

El desbordamiento en la resta puede detectarse mediante el examen de los signos del minuendo y del sustraendo *complementado*, utilizando la misma regla que en la suma. O, empleando la técnica de los ejemplos anteriores, se pueden observar los acarreos de entrada y de salida de la posición del signo y detectar el desbordamiento, independientemente de los signos de entrada y salida, usando de nueva cuenta la misma regla de la suma.

Un intento de hacer negativo el número negativo "extra" da como resultado un desbordamiento de acuerdo con las reglas anteriores, cuando agregamos 1 en el proceso de complementación:

$$-(-8) = -1000 = \begin{array}{r} 0111 \\ +\ 0001 \\ \hline 1000 \end{array} = -8$$

No obstante, este número todavía puede ser utilizado en sumas y restas mientras que el resultado final no exceda el intervalo del número:

					$1 - c_{ent}$
+4	0100		−3	1101	1101
+ −8	+ 1000		− −8	− 1000	+ 0111
−4	1100		+5		10101

2.6.5 Números binarios sin signo y complemento a dos

Puesto que los números de complemento a dos se suman y restan siguiendo los mismos algoritmos de suma y resta binaria básicas que los números sin signo de la misma longitud, una computadora u otro sistema digital puede usar el mismo circuito sumador para manejar números de ambos tipos. Sin embargo, los resultados deben ser

interpretados de manera diferente, dependiendo de si el sistema está tratando con números con signo (por ejemplo, de –8 hasta +7) o con números sin signo (por ejemplo, de 0 hasta 15).

números con signo vs. números sin signo

Presentamos una representación gráfica del sistema de complemento a dos de 4 bits en la figura 2-3. Podemos volver a etiquetar esa figura como se ilustra en la figura 2-4 para obtener una representación de los números sin signo de 4 bits. Las combinaciones binarias ocupan las mismas posiciones sobre la rueda, y un número se suma moviendo la flecha un número correspondiente de posiciones en el sentido de giro de las manecillas del reloj, y se resta desplazando la flecha en sentido contrario al giro de las manecillas del reloj.

Puede verse cómo una operación de suma excede el intervalo del sistema numérico sin signo de 4 bits en la figura 2-4, cuando la flecha se mueve en el sentido de giro de las manecillas del reloj a través de la discontinuidad entre 0 y 15. En este caso se dice que ocurre un *acarreo* hacia afuera de la posición del bit más significativo.

acarreo

Asimismo una operación de resta excede el intervalo del sistema numérico si la flecha se mueve en sentido contrario al giro de las manecillas del reloj, a través de la discontinuidad. En este caso se dice que se presenta un *traslado o préstamo* fuera de la posición del bit más significativo.

traslado o préstamo

De la figura 2-4 también es evidente que podemos restar un número sin signo n al contar *en el sentido de las manecillas del reloj* $16 - n$ posiciones. Esto equivale a *sumar* el complemento a dos de 4 bits de n. La resta produce un préstamo si la suma correspondiente del complemento a dos *no* produce un acarreo.

En resumen, en la suma sin signo, el acarreo o préstamo en la posición del bit más significativo indica un resultado fuera de intervalo. En el caso con signo que considera la suma de complemento a dos, la condición de desbordamiento que se definió anteriormente indica un resultado fuera de intervalo. El acarreo desde la posición del bit más significativo es irrelevante en la suma con signo, ya que el desbordamiento puede o no puede ocurrir independientemente de que se presente o no un acarreo.

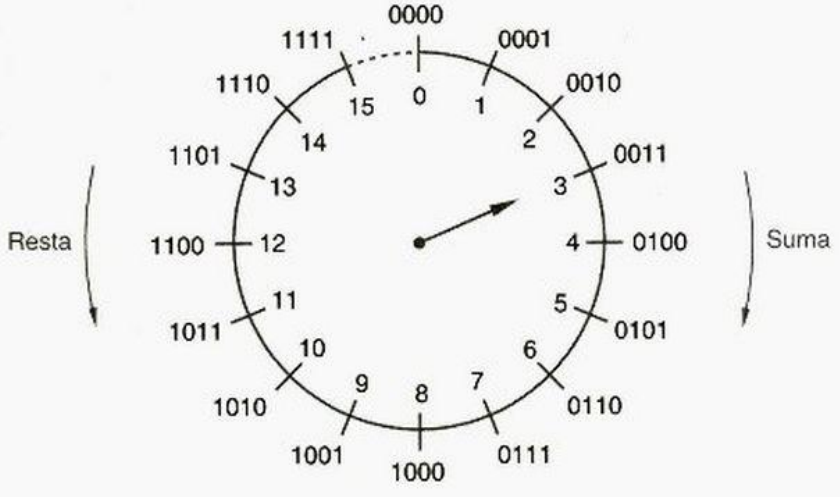


Figura 2-4
Una representación de conteo modular de números sin signo de 4 bits.

*2.7 Suma y resta de complemento a uno

Si analiza la tabla 2-6 encontrará la explicación de la regla que indica cómo sumar números de complemento a uno. Si comenzamos en 1000_2 (-7_{10}) y contamos en forma ascendente, obtenemos un número sucesivo de complemento a unos al sumar 1 al anterior, *excepto* en la transición de 1111_2 (0 negativo) a 0001_2 ($+1_{10}$). Para mantener el conteo apropiado, debemos agregar 2 en vez de 1 dondequiera que nuestro conteo rebase 1111_2. Esto sugiere una técnica para sumar números de complemento a uno: efectuar una suma binaria estándar, pero agregar un 1 extra cada vez que rebasemos 1111_2. El conteo que rebasa 1111_2 durante una suma puede detectarse al observar el acarreo de salida de la posición de signo. Por lo tanto, la regla para sumar números de complemento a uno puede establecerse de manera muy simple:

suma de complemento a uno

- Realice una suma binaria estándar; si hay un acarreo de salida de la posición de signo, agregue 1 al resultado.

acarreo de redondeo final

Esta regla se conoce a menudo como *acarreo de redondeo final* (*end-around carry*). A continuación se presentan varios ejemplos de suma de complemento a uno; los últimos tres incluyen un acarreo de redondeo final:

+3	0011	+4	0100	+5	0101
+ +4	+ 0100	+ −7	+ 1000	+ −5	+ 1010
+7	0111	−3	1100	−0	1111
−2	1101	+6	0110	−0	1111
+ −5	+ 1010	+ −3	+ 1100	+ −0	+ 1111
−7	**1**0111	+3	**1**0010	−0	**1**1110
	+ **1**		+ **1**		+ **1**
	1000		0011		1111

Siguiendo la regla de suma de dos pasos, la adición de un número y su complemento a uno produce un 0 negativo. De hecho, una operación de suma que utiliza esta regla nunca producirá un 0 positivo, a menos que ambos sumandos sean 0 positivos.

resta de complemento a unos

Como sucede en el complemento a dos, la manera más fácil de hacer la resta de complemento a uno es complementar el sustraendo y sumar. Las reglas de desbordamiento para la suma y resta del complemento a uno son las mismas que para el complemento a dos.

La tabla 2-7 resume las reglas que presentamos en ésta y las secciones anteriores para la negación, suma y resta en sistemas numéricos binarios.

Tabla 2-7 Resumen de las reglas de suma y resta para números binarios.

Sistema numérico	*Reglas de la suma*	*Reglas de la negación*	*Reglas de la resta*
Sin signo	Sumar los números. El resultado se encuentra fuera de intervalo si se presenta un acarreo fuera del MSB.	No se aplican.	Reste el sustraendo del minuendo. El resultado se encuentra fuera de intervalo si se presenta un préstamo fuera del MSB.
Magnitud con signo	(mismo signo) Sume las magnitudes; ocurre un desbordamiento si se presenta un acarreo fuera del MSB; el resultado tiene el mismo signo. (signos opuestos) Reste la magnitud más pequeña de la más grande; el desbordamiento es imposible; el resultado tiene el signo de la más grande.	Cambie el bit de signo del número.	Cambie el bit de signo del sustraendo y proceda como en la suma.
Complemento a dos	Sumar, ignorando cualquier acarreo de salida del MSB. Ocurre un desbordamiento si los acarreos de entrada y de salida del MSB son diferentes.	Complemente todos los bits del número; agregue 1 al resultado.	Complemente todos los bits del sustraendo y sume al minuendo con un acarreo inicial de 1.
Complemento a unos	Sume; si hay un acarreo fuera del MSB, agregue 1 al resultado. Se presenta un desbordamiento si los acarreos de entrada y de salida del MSB son diferentes.	Complemente todos los bits del número.	Complemente todos los bits del sustraendo y proceda como en el caso de la suma.

*2.8 Multiplicación binaria

multiplicación con corrimiento y suma

multiplicación binaria sin signo

En la escuela primaria aprendimos a multiplicar mediante la suma de una lista de multiplicandos trasladados, que se calculaban de acuerdo con los dígitos del multiplicador. Se puede utilizar el mismo método para obtener el producto de dos números binarios sin signo. La formación de los multiplicandos trasladados es trivial en la multiplicación binaria, puesto que los únicos valores posibles de los dígitos del multiplicador son 0 y 1. A continuación presentamos un ejemplo:

```
   11          1011   multiplicando
 × 13        ×  1101   multiplicador
 ----        --------
   33          1011
  11          0000    multiplicandos desplazados
 ----        1011
  143        1011
             --------
             10001111  producto
```

producto parcial

En vez de listar todos los multiplicandos trasladados y posteriormente sumar, en un sistema digital es más conveniente sumar cada multiplicando trasladado como si fuera creado para un *producto parcial*. Aplicando esta técnica al ejemplo anterior, se utilizan cuatro sumas y productos parciales para multiplicar números de 4 bits:

11	1011	multiplicando
× 13	× 1101	multiplicador
	0000	producto parcial
	1011	multiplicando desplazado
	01011	producto parcial
	0000↓	multiplicando desplazado
	001011	producto parcial
	1011↓↓	multiplicando desplazado
	0110111	producto parcial
	1011↓↓↓	multiplicando desplazado
	10001111	producto

En general, cuando multiplicamos un número de n bits por un número de m bits, para expresar el producto resultante se requieren como máximo $n + m$ bits. El algoritmo de desplazamiento y suma requiere m productos y sumas parciales para obtener el resultado, pero la primera suma es trivial, puesto que el primer producto parcial es cero. Aunque el primer producto parcial tiene solamente n bits significativos, después de cada paso de suma, el producto parcial gana un bit significativo más, ya que cada suma puede producir un acarreo. Al mismo tiempo, cada paso produce un bit de producto parcial adicional, comenzando con el que está más a la derecha y continuando hacia la izquierda, que no cambia. El algoritmo de desplazamiento y suma puede realizarse mediante un circuito digital que incluirá un registro de corrimiento, un sumador y la lógica de control, como se verá en la sección 8.7.2.

multiplicación con signo

La multiplicación de números con signo puede realizarse aplicando la multiplicación sin signo y las reglas de escuela habituales: realice una multiplicación sin signo de las magnitudes y haga el producto positivo si los operandos tienen el mismo signo, negativo si tienen signos diferentes. Esto es muy conveniente en sistemas de magnitud con signo, puesto que el signo y la magnitud están separados.

multiplicación de complemento a dos

En el sistema de complemento a dos, la obtención de la magnitud de un número negativo y la conversión a negativo del producto sin signo son operaciones poco triviales. Esto nos conduce a buscar una manera más eficiente de realizar la multiplicación de complemento a dos, la cual se describe a continuación.

Conceptualmente, la multiplicación sin signo se realiza por medio de una secuencia de sumas sin signo de los multiplicandos trasladados; en cada paso, el desplazamiento del multiplicando corresponde a la ponderación del bit multiplicador. Los bits en un número de complemento a dos tienen las mismas ponderaciones que en un número sin signo, excepto para el MSB, que tiene una ponderación negativa (véase la sección 2.5.4). De este modo, podemos realizar la multiplicación de complemento a dos mediante una secuencia de sumas de complemento a dos (de multiplicandos desplazados), excepto para el último paso, donde el multiplicando trasladado (que corresponde al MSB

del multiplicador) debe convertirse a negativo antes de sumarlo al producto parcial. Nuestro ejemplo anterior se repite a continuación, en este caso el multiplicador y el multiplicando aparecen como números de complemento a dos:

−5	1011	multiplicando
× −3	× 1101	multiplicador
	00000	producto parcial
	11011	multiplicando desplazado
	111011	producto parcial
	00000↓	multiplicando desplazado
	1111011	producto parcial
	11011↓↓	multiplicando desplazado
	11100111	producto parcial
	00101↓↓↓	multiplicando convertido a negativo y desplazado
	00001111	producto

El manejo de los MSB es algo delicado porque ganamos un bit significativo en cada paso y trabajamos con números que tienen signo. Por consiguiente, antes de sumar cada multiplicando trasladado y producto parcial de k-bits, los cambiamos a $k + 1$ bits significativos por extensión de signo, como se muestra en negritas en el desarrollo de la operación. Cada suma resultante tiene $k + 1$ bits y se ignora cualquier acarreo fuera del MSB de la suma del $k + 1$ bit.

*2.9 División binaria

división de desplazamiento y resta

división sin signo

El algoritmo más simple de división binaria se basa en el método de desplazamiento y resta que aprendimos cuando estudiamos en la primaria. La tabla 2-8 proporciona ejemplos de este método para números binarios y decimales sin signo. En ambos casos, comparamos mentalmente el dividendo reducido con múltiplos del divisor para determinar cuál múltiplo del divisor desplazado se debe restar. En el caso decimal, primero elegimos 11 como el mayor múltiplo de 11 menor que 21 y luego elegimos 99 como el

19	10011	cociente	
11)217	1011)11011001	dividendo	
11	1011	divisor desplazado	
107	0101	dividendo reducido	
99	0000	divisor desplazado	
8	1010	dividendo reducido	
	0000	divisor desplazado	
	10100	dividendo reducido	
	1011	divisor desplazado	
	10011	dividendo reducido	
	1011	divisor desplazado	
	1000	residuo	

Tabla 2-8 Ejemplo de división larga.

mayor múltiplo menor que 107. En el caso binario, la elección es más sencilla, puesto que las únicas opciones son cero y el mismo divisor.

Los métodos de división para números binarios son un tanto complementarios a los métodos de multiplicación binaria. Un típico algoritmo de división toma un dividendo de $(n + m)$ bits y un divisor de n bits, para producir un cociente de m bits y un residuo de n bits. El *desbordamiento* de la división se presentará cuando el divisor sea cero o el cociente necesita más de m bits para expresarse. En la mayoría de los circuitos de división de las computadoras, $n = m$.

desbordamiento de división

división con signo

La división de números con signo puede llevarse a cabo aplicando la división sin signo y las reglas habituales: efectuar una división sin signo de las magnitudes y hacer el cociente positivo si los operandos tienen el mismo signo, y negativo si tienen signos diferentes. Se debería dar al residuo el mismo signo que al dividendo. Como en el caso de la multiplicación, existen técnicas especiales para realizar directamente la división sobre números de complemento a dos; con cierta frecuencia se implementan estas técnicas en circuitos de división para computadoras (véanse las referencias).

2.10 Códigos binarios para números decimales

Aun cuando los números binarios son los más apropiados para los cálculos internos de un sistema digital, la mayoría de la gente prefiere trabajar con números decimales. Como resultado, las interfaces externas de un sistema digital pueden leer o presentar números decimales, y en realidad algunos dispositivos digitales procesan directamente números decimales.

La necesidad humana de representar los números decimales no cambia la naturaleza básica de los circuitos electrónicos digitales; éstos procesan las señales que pueden tener uno de dos estados que llamamos 0 y 1. Por tanto, un número decimal se representa en un sistema digital mediante una cadena de bits, donde las diferentes combinaciones de los valores de bits en la cadena representan diferentes números decimales. Por ejemplo, si empleamos una cadena de 4 bits para representar un número decimal, podemos asignar la combinación de bits 0000 al dígito decimal 0, 0001 al 1, 0010 al 2, y así sucesivamente.

Un conjunto de cadenas de n bits en el cual diferentes cadenas de bits representan diferentes números u otras cosas se denomina *código*. Una combinación particular de valores de n bits se conoce como *palabra de código*. Como veremos en los ejemplos de códigos decimales en esta sección, puede o no existir una relación aritmética entre los valores de bits en una palabra de código y lo que representa. Por eso, un código que utiliza cadenas de n bits no necesita contener palabras de código válidas de 2^n.

código

palabra de código

Como mínimo se necesitan cuatro bits para representar los diez dígitos decimales. Existen miles y miles de millones de maneras diferentes para elegir 10 palabras de código de 4 bits; sin embargo, la tabla 2-9 muestra los códigos decimales más comunes.

Quizás el código decimal más "natural" sea el *decimal codificado en binario (BCD)*, el cual codifica los dígitos del 0 al 9 mediante sus representaciones binarias sin signo de 4 bits, desde 0000 hasta 1001. Las palabras de código restantes, de 1010 a 1111, no se utilizan. Las conversiones entre las representaciones BCD y decimal son triviales, implican la sustitución directa de cuatro bits por cada dígito decimal. Algunos programas de computadora colocan dos dígitos BCD en un byte de 8 bits en la *representación BCD*

decimal codificado en binario (BCD)

representación BCD empaquetada

Tabla 2-9 Códigos decimales.

Dígito decimal	*BCD (8421)*	*2421*	*Exceso-3*	*Biquinario*	*1 de 10*
0	0000	0000	0011	0100001	1000000000
1	0001	0001	0100	0100010	0100000000
2	0010	0010	0101	0100100	0010000000
3	0011	0011	0110	0101000	0001000000
4	0100	0100	0111	0110000	0000100000
5	0101	1011	1000	1000001	0000010000
6	0110	1100	1001	1000010	0000001000
7	0111	1101	1010	1000100	0000000100
8	1000	1110	1011	1001000	0000000010
9	1001	1111	1100	1010000	0000000001
		Palabras de código sin utilizar			
	1010	0101	0000	0000000	0000000000
	1011	0110	0001	0000001	0000000011
	1100	0111	0010	0000010	0000000101
	1101	1000	1101	0000011	0000000110
	1110	1001	1110	0000101	0000000111
	1111	1010	1111	...	...

empaquetada; así, un byte puede representar los valores desde 0 hasta 99 en oposición al intervalo de 0 a 255 para un número binario normal de 8 bits sin signo. Se pueden obtener los números BCD con cualquier cantidad de dígitos mediante el uso de un byte para cada par de dígitos.

Como sucede con los números binarios, existen muchas representaciones posibles de los números BCD negativos. Los números BCD con signo tienen una posición de dígito extra para el signo. Tanto la representación de magnitud con signo como las

COEFICIENTES BINOMIALES

La cantidad de formas diferentes para elegir m elementos de un conjunto de n elementos está determinada por un *coeficiente binomial*, denotado $\binom{n}{m}$, cuyo valor es $\frac{(n)!}{m! \cdot (n-m!)}$. Para un código decimal de 4 bits, existen $\binom{16}{10}$ formas diferentes de extraer 10 de 16 palabras de código de 4 bits, y existen 10! formas de asignar cada selección diferente a los 10 dígitos. De modo que tenemos $\frac{16!}{10! \cdot 6!} \cdot 10!$ o bien, 29,059,430,400 códigos decimales diferentes de 4 bits.

representaciones del complemento de 10 son populares. En BCD de magnitud con signo, la codificación de la cadena de bit de signo es arbitraria; en el complemento de 10, el dato 0000 indica el signo más y 1001 indica el signo menos.

suma BCD

La suma de los dígitos BCD es semejante la adición de números binarios sin signo de 4 bits, excepto que debe hacerse una corrección si un resultado excede 1001. El resultado se corrige agregando 6; como se muestra en los siguientes ejemplos:

5	0101		4	0100	
+ 9	+ 1001		+ 5	+ 0101	
14	1110		9	1001	
	+ 0110	— **corrección**			
10 + 4	**1** 0100				
8	1000		9	1001	
+ 8	+ 1000		+ 9	+ 1001	
−16	**1** 0000		18	**1** 0010	
	+ 0110	— **corrección**		+ 0110	— **corrección**
10 + 6	**1** 0110		10 + 8	**1** 1000	

Nótese que la suma de dos dígitos BCD produce un acarreo en la siguiente posición de dígito, si la suma binaria inicial o la suma de factor de corrección produce un acarreo. Muchas computadoras realizan aritmética de BCD empaquetado aplicando instrucciones especiales que manejan la corrección de acarreo en forma automática.

código ponderado

El decimal codificado en binario es un *código ponderado,* puesto que cada dígito decimal puede obtenerse a partir de su palabra de código asignando un peso fijo a cada bit de palabra de código. Los pesos para los bits BCD son 8, 4, 2 y 1, y por esta razón el

código 8421

código se denomina en ocasiones *código 8421.* Otro conjunto de pesos da como resulta-

código 2421

do el *código 2421* que se indica en la tabla 2-9. Este código tiene la ventaja de que es

código autocomplementado

autocomplementado, es decir, se puede obtener la palabra de código para el complemento a nueve de cualquier dígito al complementar los bits individuales de la palabra de código del dígito.

código de exceso 3

La tabla 2-9 muestra otro código autocomplementado, el *código de exceso 3.* Aunque este código no está ponderado, tiene una relación aritmética con el código BCD (la palabra de código para cada dígito decimal es la correspondiente palabra de código BCD más 0011_2). Como las palabras del código siguen una secuencia de conteo binaria estándar, pueden hacerse fácilmente contadores binarios estándar para contar en el código de exceso 3, como mostraremos en la figura 8-37 (pág. 700).

código biquinario

Los códigos decimales pueden tener más de cuatro bits; por ejemplo, el *código biquinario* de la tabla 2-9 utiliza siete. Los primeros dos bits en una palabra de código indican si el número se encuentra en el intervalo 0-4 o 5-9, mientras que los cinco últimos indican cuál de los cinco números en el intervalo seleccionado está representado.

Una ventaja potencial que se obtiene al utilizar más del número mínimo de bits en un código es la propiedad de detección de errores. En el código biquinario, si cualquier bit en una palabra de código se cambia accidentalmente al valor opuesto, la palabra de

código resultante no representará un dígito decimal y por consiguiente se puede señalar como un error. Fuera de 128 posibles palabras de código de 7 bits, solamente 10 son válidas y reconocidas como dígitos decimales; el resto se puede señalar como error, si es que aparecen.

código 1 de 10

Un *código 1 de 10*, tal como el que se muestra en la última columna de la tabla 2-9, es la codificación más escasa para dígitos decimales, utilizando 10 de 1024 posibles palabras de código de 10 bits.

2.11 Código Gray

En las aplicaciones electromecánicas de los sistemas digitales (tales como herramientas mecánicas, sistemas de frenado para automóviles y fotocopiadoras) a veces es necesario que un sensor de entrada produzca un valor digital que indique una posición mecánica. Por ejemplo, la figura 2-5 es un esquema conceptual de un disco de codificación y un conjunto de contactos que producen uno de ocho valores codificados en binario de 3 bits, dependiendo de la posición rotacional del disco. Las áreas oscuras del disco se conectan a una fuente de señal que corresponde al 1 lógico y las áreas claras no se conectan, lo que los contactos interpretan como un 0 lógico.

El codificador de la figura 2-5 tiene un problema cuando el disco se coloca en ciertas fronteras entre las regiones. Por ejemplo, considere la frontera entre las regiones 001 y 010 del disco; dos de los bits codificados cambian aquí. ¿Qué valor producirá el codificador si el disco se posiciona justo sobre la frontera teórica? Puesto que nos encontramos sobre el borde, tanto 001 como 010 son aceptables. Sin embargo, ya que el ensamble mecánico no es perfecto, los dos contactos de la derecha pueden tocar una región "1", dando una lectura incorrecta de 011. De igual forma, se puede obtener una lectura de 000. En general, esta clase de problema puede presentarse en cualquier frontera donde cambia más de un bit. Los peores problemas ocurren cuando cambian los tres bits, como en las fronteras 000-111 y 011-100.

El problema del disco codificado puede resolverse mediante la creación de un código digital en el cual solamente cambie un bit entre cada par de palabras de código sucesivas. Un código de esta clase se denomina *código Gray*; la tabla 2-10 muestra un código Gray de 3 bits. Hemos rediseñado el disco codificado usando este código como

código Gray

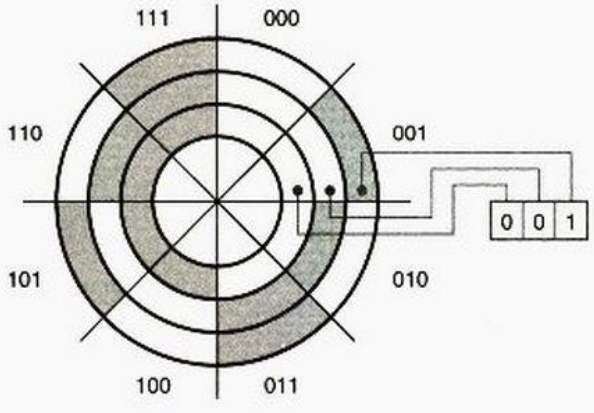


Figura 2-5
Disco mecánico de codificación que utiliza un código binario de 3 bits.

Tabla 2-10
Una comparación de código binario de 3 bits y código Gray.

Número decimal	*Código binario*	*Código Gray*
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

se muestra en la figura 2-6. Solamente cambia un bit del nuevo disco en cada borde, de modo que las lecturas de las líneas fronterizas nos dan un valor que está en cualquiera de los lados del borde.

Existen dos maneras convenientes de construir un código Gray con cualquier número deseado de bits. El primer método está basado en el hecho de que el código Gray es un *código reflejado*; puede ser definido (y construido) de manera recursiva utilizando las reglas siguientes:

código reflejado

1. Un código Gray de 1 bit tiene dos palabras de código, 0 y 1.
2. Las primeras 2^n palabras de código de un código Gray de $(n + 1)$ bits son iguales a las palabras de código de un código Gray de n bits, escritas en orden con un 0 principal agregado.
3. Las últimas 2^n palabras de código de un código Gray de $(n + 1)$ bits son iguales a las palabras de código de un código Gray de n bits, pero escritas en orden inverso con un 1 principal agregado.

Si trazamos una línea entre los renglones 3 y 4 de la tabla 2-10, podemos ver que las reglas 2 y 3 son verdaderas para el código Gray de 3 bits. Por supuesto, para construir un código Gray de n bits para un valor arbitrario de n con este método, también debemos construir un código Gray para cada longitud más pequeña que n.

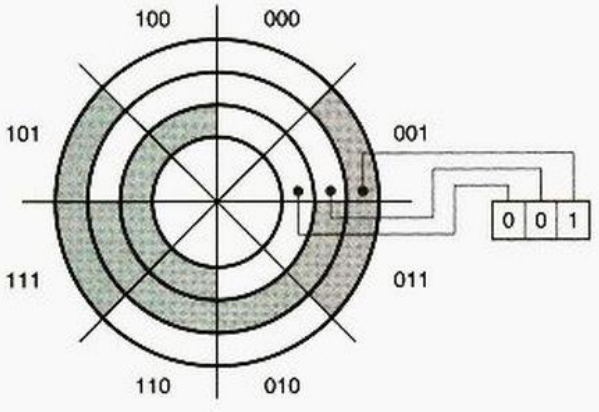


Figura 2-6
Un disco mecánico de codificación que utiliza un código Gray de 3 bits.

El segundo método nos permite derivar una palabra de código (del código Gray) de n bits directamente de la correspondiente palabra de código binaria de n bits:

1. Los bits de una palabra de código binaria de n bits o código Gray se numeran de derecha a izquierda, desde 0 hasta $n - 1$.
2. El bit i de una palabra de código del código Gray es 0 si los bits i e $i + 1$ de la correspondiente palabra de código binario son los mismos, si no el bit i es 1. (Cuando $i + 1 = n$, el bit n de la palabra de código binario se considera que es 0.)

De nueva cuenta, el examen de la tabla 2-10 muestra que esto es verdadero para el código Gray de 3 bits.

*2.12 Códigos de carácter

Como mostramos en la sección precedente, una cadena de bits no necesita representar un número y en realidad la mayor parte de la información que procesan las computadoras no es numérica. El tipo más común de datos no numéricos es el *texto*, cadenas de caracteres de algún conjunto de caracteres. Cada carácter es representado en la computadora por una cadena de bits de acuerdo a una convención establecida.

texto

El código de caracteres más comúnmente utilizado es el *ASCII* (pronunciado *ASSKI, American Standard Code for Information Interchange*), el Código Estandar Americano para Intercambio de Información, por sus siglas en inglés. El ASCII representa cada carácter con una cadena de 7 bits, y produce un total de 128 caracteres diferentes, los cuales se indican en la tabla 2-11. El código contiene el alfabeto en mayúsculas y minúsculas, números, signos de puntuación y diversos caracteres de control no imprimibles. De este modo, la cadena de texto "Yeccch!" está representada por una lista de aspecto bastante inocuo de siete números de 7 bits:

ASCII

1011001 1100101 1100011 1100011 1100011 1101000 0100001

2.13 Códigos para acciones, condiciones y estados

Los códigos que hemos descrito hasta ahora se emplean generalmente para representar cosas que usted probablemente consideraría como "datos" (cosas como números, posiciones y caracteres). Los programadores saben que docenas de diferentes tipos de datos pueden ser utilizados en un solo programa de computadora.

En el diseño de sistemas digitales, con frecuencia hallamos aplicaciones sin datos donde una cadena de bits debe emplearse para controlar una acción, para señalar una condición, o para representar el estado actual del hardware. Probablemente el tipo de código más comúnmente empleado para una aplicación de esta clase es un simple código binario. Si hay n diferentes acciones, condiciones o estados, podemos representarlos con un código binario de b bits con $b = \lceil \log_2 n \rceil$ bits. (Los símbolos $\lceil\ \rceil$ indican la *función techo*, el entero más pequeño mayor o igual que la cantidad encerrada entre ellos. De este modo, b es el entero más pequeño tal que $2^b \geq n$.)

$\lceil\ \rceil$

función techo

Por ejemplo, considere un simple controlador de semáforos. Las señales en la intersección de una calle en dirección norte-sur (N-S) y una de este a oeste (E-W) pueden estar en cualquiera de los seis estados que se indican en la tabla 2-12. Estos estados pueden estar codificados en tres bits, como se muestra en la última columna de la tabla. Solamente seis de las ocho posibles palabras de código de 3 bits se utilizan, y la asignación de las seis palabras de

Tabla 2-11 Código Estándar Americano para Intercambio de Información (ASCII), estándar No. X3.4-1968 del American National Standards Institute.

		$b_6b_5b_4$ (columna)							
$b_3b_2b_1b_0$	Renglón (hex)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	'	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS		=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Códigos de control

NUL	Nulo (Null)	DLE	Escape de enlace de datos (Data link escape)
SOH	Comienzo de encabezado (Start of heading)	DC1	Control de dispositivo 1 (Device control 1)
STX	Comienzo de texto (Start of text)	DC2	Control de dispositivo 2 (Device control 2)
ETX	Fin de texto (End of text)	DC3	Control de dispositivo 3 (Device control 3)
EOT	Fin de transmisión (End of transmission)	DC4	Control de dispositivo 4 (Device control 4)
ENQ	Pregunta (Enquiry)	NAK	Admisión negativa (Negative acknowledge)
ACK	Admisión (Acknowledge)	SYN	Sincronía (Synchronize)
BEL	Campana (Bell)	ETB	Fin de bloque transmitido (End transmitted block)
BS	Retroceso (Backspace)	CAN	Cancelación (Cancel)
HT	Tabulador horizontal (Horizontal tab)	EM	Fin del medio (End of medium)
LF	Avance de línea (Line feed)	SUB	Sustitución (Substitute)
VT	Tabulador vertical (Vertical tab)	ESC	Escape (Escape)
FF	Avance de página (Form feed)	FS	Separador de archivo (File separator)
CR	Retorno de carro (Carriage return)	GS	Separador de grupo (Group separator)
SO	Corrimiento hacia fuera (Shift out)	RS	Separador de registro (Record separator)
SI	Corrimiento hacia dentro (Shift in)	US	Separador de unidad (Unit separator)
SP	Espacio (Space)	DEL	Eliminación o borrado (Delete or rubout)

Tabla 2-12 Estados en un controlador de semáforos.

	Luces						
Estado	**Verde N-S**	**Amarillo N-S**	**Rojo N-S**	**Verde E-W**	**Amarillo E-W**	**Rojo E-W**	**Palabra de código**
N-S siga	ENCEN-DIDO	apagado	apagado	apagado	apagado	ENCEN-DIDO	000
N-S espere	apagado	ENCEN-DIDO	apagado	apagado	apagado	ENCEN-DIDO	001
N-S alto	apagado	apagado	ENCEN-DIDO	apagado	apagado	ENCEN-DIDO	010
E-W siga	apagado	apagado	ENCEN-DIDO	ENCEN-DIDO	apagado	apagado	100
E-W espere	apagado	apagado	ENCEN-DIDO	apagado	ENCEN-DIDO	apagado	101
E-W alto	apagado	apagado	ENCEN-DIDO	apagado	apagado	ENCEN-DIDO	110

código elegidas a los estados es arbitraria, de modo que muchas otras codificaciones son posibles. Un diseñador digital experimentado elige una codificación particular para minimizar el costo del circuito o para optimizar algún otro parámetro (como el tiempo de diseño: no hay necesidad de intentar miles y miles de millones de posibles codificaciones).

Otra aplicación del código binario se ilustra en la figura 2-7(a). Aquí, tenemos un sistema con n dispositivos, cada uno de los cuales puede realizar una acción en particular. Las características de los dispositivos son tales que pueden ser habilitados para funcionar solamente uno a la vez. La unidad de control produce una palabra de "selección de dispositivo" codificada en binario con $\lceil \log_2 n \rceil$ bits para indicar cuál dispositivo se encuentra habilitado en cualquier tiempo. La palabra de código "selectora de dispositivo" se aplica a cada dispositivo, a su vez ésta se compara con la "ID propia del dispositivo" para determinar si se encuentra habilitado. Aunque sus palabras de código tienen el mínimo número de bits, un código binario no siempre es la mejor elección para codificar acciones, condiciones o estados. La figura 2-7(b) muestra cómo controlar n dispositivos con un *código 1 de n*, un código de n bits en el cual las palabras de código válidas tienen un bit igual a 1 y el resto de los bits son iguales a 0. Cada bit de la palabra del código 1 de n se conecta directamente a la entrada de habilitación del dispositivo correspondiente. Esto simplifica el diseño de los dispositivos, puesto que ellos ya no tienen ID de dispositivo; solamente necesitan un bit de entrada de "habilitación".

código 1 de n

Las palabras de código de un código 1 de 10 se indican en la tabla 2-9. En ocasiones una palabra "llena" de ceros puede incluirse en un código 1 de n, para indicar que ningún dispositivo se encuentra seleccionado. Otro código común es un *código 1 de n invertido*, en el cual las palabras de código válidas tienen un bit 0 y el resto de los bits son iguales a 1.

código 1 de n invertido

En sistemas complejos, puede emplearse una combinación de las técnicas de codificación. Por ejemplo, considere un sistema similar al que se muestra en la figura 2-7(b), en el que cada uno de los n dispositivos contiene hasta s subdispositivos. La unidad de control produciría una palabra de código de selección de dispositivo con un campo codi-

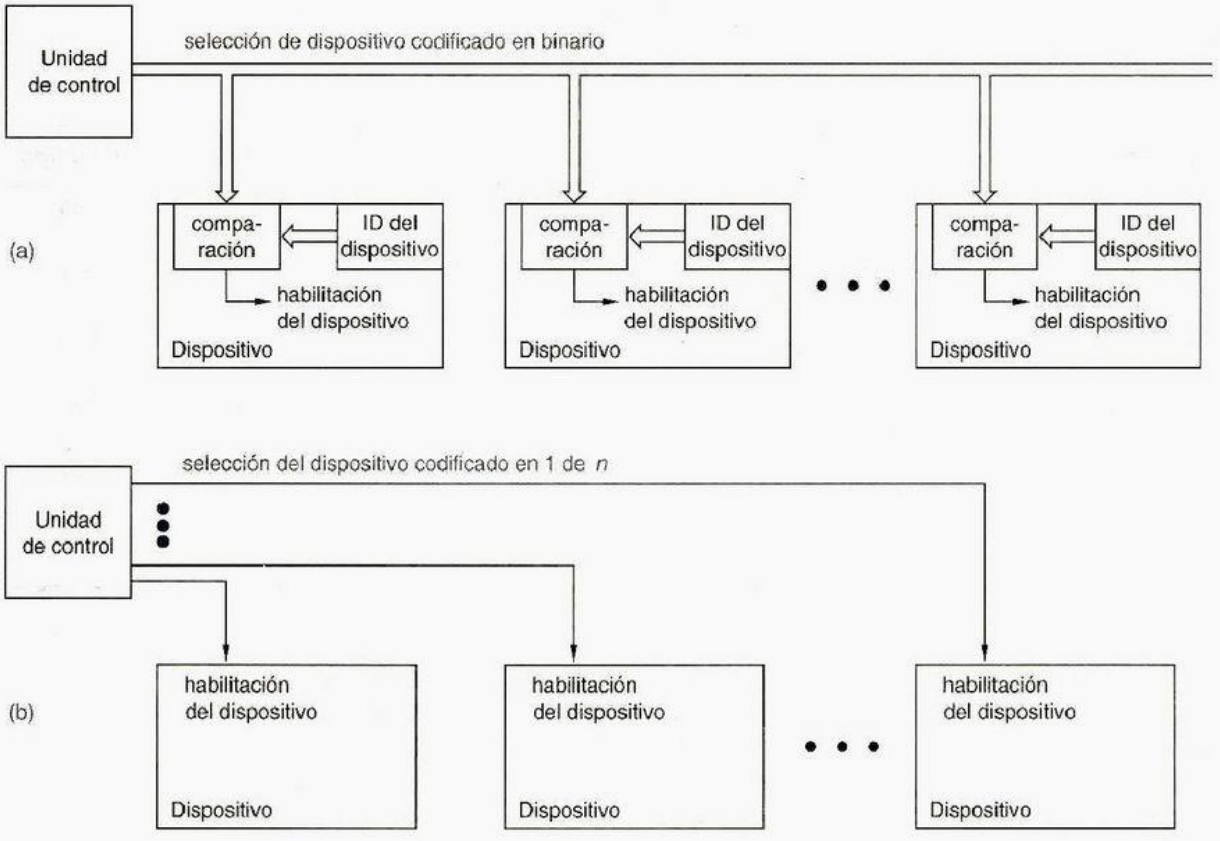


Figura 2-7 Estructura de control para un sistema digital con n dispositivos: a) que utiliza un código binario; b) que utiliza un código 1 de n.

ficado de 1 de n para seleccionar un dispositivo, y un campo codificado en binario de $\lceil \log_2 s \rceil$ bits para seleccionar uno de los s subdispositivos del dispositivo seleccionado.

código m de n

Un *código m de n* es una generalización del código 1 de n, en el cual las palabras de código válidas tienen m bits iguales a 1 y el resto de los bits son iguales a 0. Una palabra de código m de n puede ser detectada con una compuerta AND de entrada m, que produce una salida 1 si todas sus entradas son 1. Esto es simple y barato de hacer, además, para la mayor parte de los valores de m, un código m de n típicamente tiene palabras de código mucho más válidas que un código 1 de n. El número total de palabras de código está dado por el coeficiente binomial $\binom{n}{m}$, el cual tiene el valor $\frac{n!}{m! \cdot (n-m)!}$. Así, un código 2 de 4 tiene 6 palabras de código válidas y un código 3 de 10 tiene 120.

código 8B10B

Una variación importante en el código m de n es el *código 8B10B* que se utiliza en el estándar Ethernet Gigabit 802.3z. Este código emplea 10 bits para representar 256 palabras de código válidas, o valor de datos de 8 bits. La gran mayoría de las palabras de código utilizan una codificación de 5 de 10. Sin embargo, puesto que $\binom{5}{10}$ es sólo 252, algunas palabras de 4 de 10 y 6 de 10 también se emplean para completar el código de una forma muy interesante; se verá más sobre esto en la sección 2.16.2.

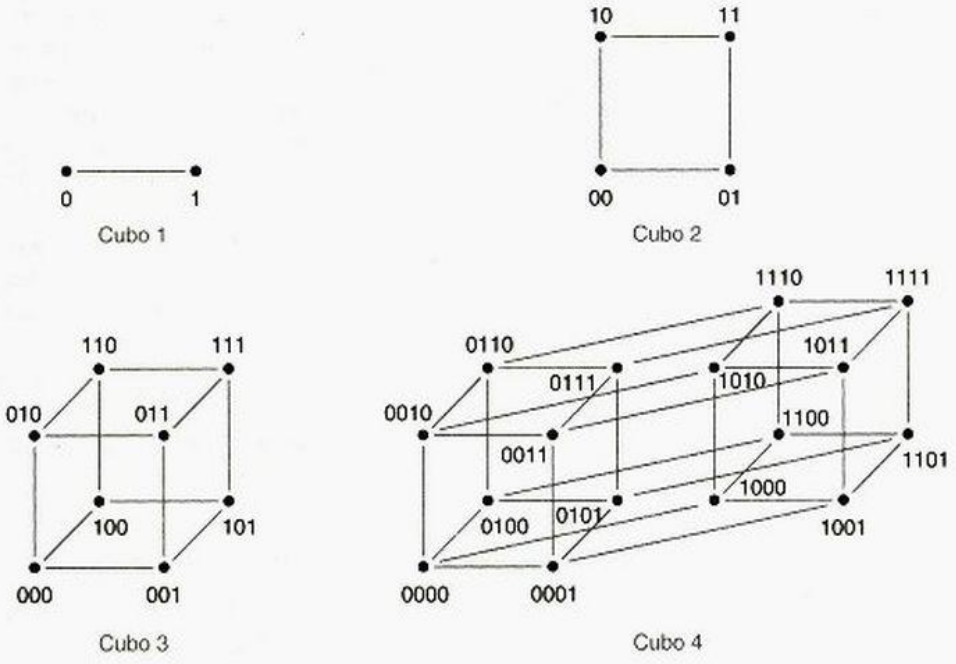


Figura 2-8
Cubos *n* para *n* = 1, 2, 3 y 4.

*2.14 Cubos *n* y distancia

Una cadena de *n* bits puede visualizarse geométricamente, como un vértice de un objeto llamado un *cubo n*. La figura 2-8 muestra cubos *n* para *n* = 1, 2, 3, 4. Un cubo *n* tiene 2^n vértices, cada uno de los cuales se encuentra etiquetado con una cadena de *n* bits. Los bordes se dibujan de manera que cada vértice sea adyacente a otros *n* vértices cuyas etiquetas difieren del vértice dado solamente en un bit. Más allá de *n* = 4, los cubos *n* son realmente difíciles de dibujar.

cubo n

Para valores razonables de *n*, los cubos *n* hacen fácil la visualización de ciertos problemas de codificación y minimización de lógica. Por ejemplo, el problema del diseño de un código Gray de *n* bits es equivalente a encontrar una trayectoria a lo largo de los bordes de un cubo *n*, una trayectoria que visita cada vértice exactamente una vez. Las trayectorias para códigos Gray de 3 y de 4 bits se muestran en la figura 2-9.

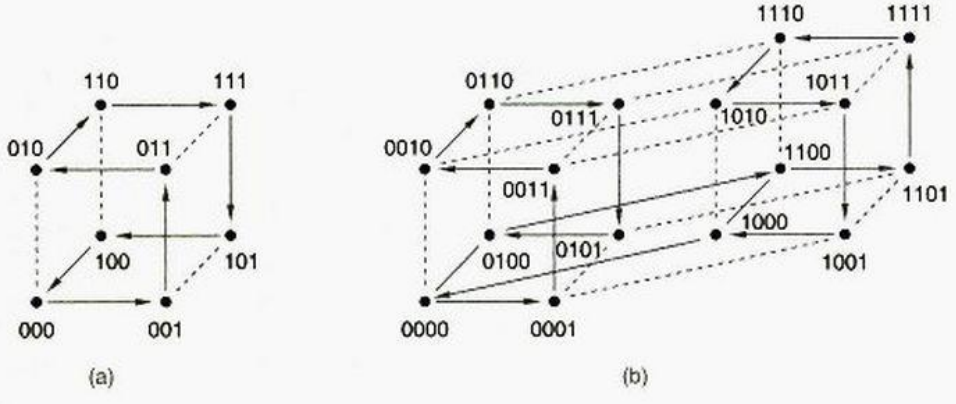


Figura 2-9
Atravesando cubos *n* en orden de código Gray: a) cubo 3; b) cubo 4.

distancia
distancia de Hamming

Los cubos también proporcionan una interpretación geométrica para el concepto de *distancia*, también llamado *distancia de Hamming*. La distancia entre dos cadenas de n bits es el número de posiciones de bit en las cuales difieren. En términos de un cubo n, la distancia es la mínima longitud de una trayectoria entre los dos vértices correspondientes. Dos vértices adyacentes tienen distancia 1; los vértices 001 y 100 en el cubo 3 tienen distancia 2. El concepto de distancia es crucial en el diseño y comprensión de códigos de detección de errores, discutidos en la siguiente sección.

subcubo m

Un *subcubo m* de un cubo n es un conjunto de 2^m vértices en el cual $n - m$ de los bits tienen el mismo valor en cada vértice y los m bits restantes se hacen cargo de todas las combinaciones restantes 2^m. Por ejemplo, los vértices (000, 010, 100, 110) forman un subcubo 2 del cubo 3. Este subcubo también puede indicarse mediante una cadena simple,

"sin importancia"

xx0, donde "x" indica que un bit en particular es del tipo "sin *importancia*"; cualquier vértice cuyos bits coincidan en las posiciones no x pertenece a este subcubo. El concepto de subcubos es particularmente útil al visualizar algoritmos que minimizan el costo de las funciones de lógica combinacional, como mostraremos en la sección 4.4.

*2.15 Códigos para detectar y corregir errores

error
falla
falla temporal
falla permanente

Un *error* en un sistema digital es la alteración de datos a partir de su valor correcto a algún otro valor. Un error es causado por una *falla* física. Las fallas pueden ser temporales o permanentes. Por ejemplo, un rayo cósmico o una partícula alfa pueden ocasionar una falla temporal de un circuito de memoria, cambiando el valor de un bit que está almacenado en ella. Dejar que un circuito se sobrecaliente o que sufra una descarga de electricidad estática puede provocar una falla permanente, de tal manera que nunca volverá a funcionar en forma correcta.

modelo de error
modelo de error independiente

Los efectos que tienen las fallas en los datos se predicen mediante *modelos de error*. El modelo de error más sencillo que consideraremos aquí, se conoce como el *modelo de error independiente*. Este modelo supone que solamente una falla física afecta a un bit de datos; se dice que los datos corrompidos solamente contienen *un error*. Varias fallas pueden ocasionar *errores* múltiples (dos o más bits con error) pero normalmente se supone que la aparición de errores múltiples es menos probable que la aparición de errores simples.

error simple
error múltiple

2.15.1 Códigos de detección de error

Recordemos de nuestras definiciones en la sección 2.10 que un código que utiliza cadenas de n bits no necesita contener 2^n palabras de código válidas; ciertamente éste es el caso para los códigos que ahora consideramos. Un *código de detección de error* tiene la propiedad de que la corrupción o confusión de una palabra de código probablemente producirá una cadena de bits que no sea una palabra de código (una *palabra de no código*).

código de detección de error
palabra de no código

Un sistema que utiliza un código de detección de error genera, transmite y almacena solamente palabras de código. De este modo, los errores en una cadena de bits pueden detectarse mediante una regla simple: si la cadena de bits es una palabra de código, se supone que es correcta; si es una palabra de no código, entonces contiene un error.

Un código de n bits y sus propiedades de detección de error bajo el modelo de error independiente se explican fácilmente en términos de un cubo n. Un código es simplemente un subconjunto de los vértices del cubo n. A fin de que el código detecte todos

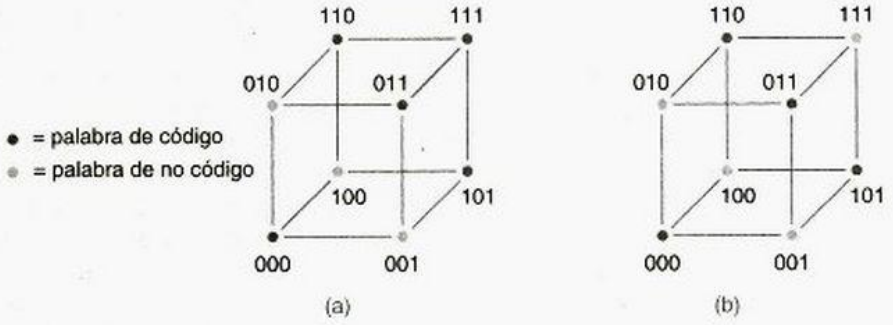


Figura 2-10
Palabras de código en dos diferentes códigos de 3 bits: a) distancia mínima = 1, no detecta todos los errores simples; b) distancia mínima = 2, detecta todos los errores simples.

los errores simples, ningún vértice de palabra de código puede estar adyacente a otro vértice de palabra de código.

Por ejemplo, la figura 2-10(a) muestra un código de 3 bits con cinco palabras de código. La palabra de código 111 se encuentra adyacente a las palabras de código 110, 011 y 101. Puesto que una falla simple podría cambiar 111 a 110, 011 o 101, este código no detecta todos los errores simples. Si hacemos 111 una palabra de no código, obtenemos un código que tiene la propiedad de detección de errores simples, como se ilustra en (b). Ningún error simple puede cambiar una palabra de código en otra.

La capacidad de un código para detectar errores simples puede establecerse en términos del concepto de distancia que se presentó en la sección anterior:

- Un código detecta todos los errores simples si la *distancia mínima* entre todos los pares posibles de palabras de código es 2.

distancia mínima

En general, necesitamos $n + 1$ bits para construir un código de detección de errores simples con 2^n palabras de código. Los primeros n bits de una palabra de código, llamados *bits de información*, pueden ser cualquiera de las 2^n cadenas de n bits. Para obtener un código de mínima distancia 2, agregamos un bit adicional, llamado *bit de paridad*, que se establece a 0 si hay un número par de unos entre los bits de información, y 1 de otro modo. Esto se ilustra en las primeras dos columnas de la tabla 2-13 para un código con tres bits de información. Una palabra de código válida de $(n + 1)$ bits tiene un número par de unos, y este código se llama *código de paridad par*. También podemos construir

bit de información
bit de paridad

código de paridad par

Tabla 2-13
Códigos de distancia 2 con tres bits de información.

Bits de información	*Código de paridad par*	*Código de paridad impar*
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

código de paridad impar

código de paridad de 1 bit

un código en el cual el número total de unos en una palabra de código válida de $(n + 1)$ bits sea impar; esto se conoce como un *código de paridad impar* y se muestra en la tercera columna de la tabla. Estos códigos también se denominan en ocasiones *códigos de paridad de 1 bit*, puesto que cada uno de ellos utiliza sólo un bit de paridad.

Los códigos de paridad de 1 bit no detectan errores de 2 bits, puesto que cambiar dos bits no afecta la paridad. Sin embargo, los códigos pueden detectar errores en cualquier número *impar* de bits. Por ejemplo, si tres bits en una palabra de código se cambian, entonces la palabra resultante tiene la paridad equivocada y es una palabra de no código. Esto no nos ayuda mucho. Bajo el modelo de error independiente, los errores de 3 bits son mucho menos probables que los errores de 2 bits, los cuales no se pueden detectar. De esta forma, prácticamente hablando, la capacidad de detección de errores de los códigos de paridad de 1 bit se detiene después de los errores de 1 bit. Se pueden utilizar otros códigos con distancia mínima mayor que 2 para detectar errores múltiples.

2.15.2 Corrección de errores y códigos de detección de errores múltiples

bits de verificación

Al utilizar más de un bit de paridad, o *bits de verificación*, de acuerdo a algunas reglas bien elegidas, podemos crear un código cuya distancia mínima sea mayor de 2. Antes de demostrar la manera en que puede hacerse esto, examinemos cómo se emplea un código de esta clase para corregir errores simples o detectar errores múltiples.

Supongamos que un código tiene una distancia mínima de 3. La figura 2-11 muestra un fragmento del cubo n para un código así. Como se muestra, existen al menos dos palabras de no código entre cada par de palabras de código. Ahora supongamos que

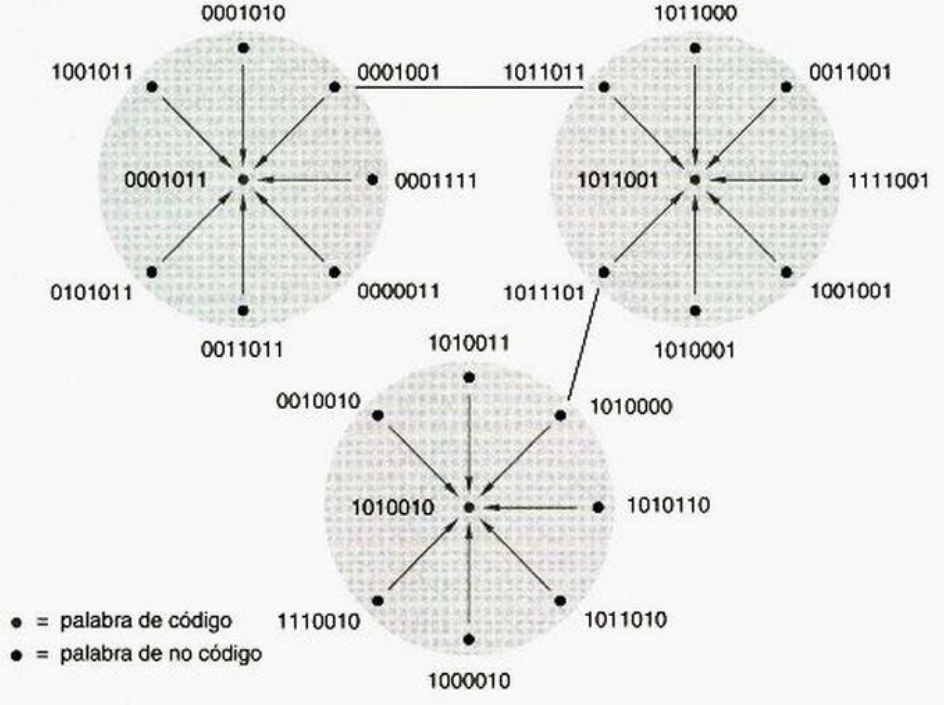


Figura 2-11
Algunas palabras de código y palabras de no código en un código de distancia 3, con 7 bits.

transmitimos palabras de código y asumamos que las fallas afectan como máximo un bit de cada palabra de código recibida. Entonces una palabra de no código recibida con un error de 1 bit estará más cerca de la palabra de código (que se transmitió originalmente) que de cualquier otra palabra de código. Por consiguiente, cuando recibimos una palabra de no código, podemos *corregir* el error cambiando la palabra de no código a la palabra de código más cercana, como indican las flechas de la figura. La decisión que implica saber cuál palabra de código se transmitió originalmente para producir una palabra que se recibió se conoce como*decodificación*, y el hardware que hace posible esto es un *decodificador* de corrección de errores.

corrección de error

decodificación
decodificador

Un código que se utiliza para corregir errores se denomina *código de corrección de errores*. En general, si un código tiene una distancia mínima de $2c + 1$, puede utilizarse para corregir errores que afecten hasta c bits ($c = 1$ en el ejemplo anterior). Si una distancia mínima de código es $2c + d + 1$, puede emplearse para corregir errores en hasta c bits y detectar errores hasta en d bits adicionales.

código de corrección de errores

Por ejemplo, la figura 2-12(a) muestra un fragmento del cubo n para un código con distancia mínima 4 ($c = 1$, $d = 1$). Los errores de un solo bit que producen palabras de no código 00101010 y 11010011 pueden corregirse. No obstante, un error que produce 10100011 no puede corregirse, porque ningún error de un solo bit puede producir esta palabra de no código, y cualquiera de los dos errores de 2 bits podría producir esta palabra. De modo que el código puede detectar un error de 2 bits, pero no puede corregirlo.

Cuando se recibe una palabra de no código, no sabemos qué palabra de código se transmitió originalmente; solamente sabemos cuál palabra de código se encuentra más cercana a lo que hemos recibido. Así, como se muestra en la figura 2-12(b), un error de 3 bits puede "corregirse". La posibilidad de hacer esta clase de equivocación puede ser aceptable si los errores de 3 bits tienen muy poca probabilidad de ocurrir. Por otro lado, si estamos interesados en los errores de 3 bits, podemos cambiar la política de decodificación para el código. En lugar de intentar corregir errores, únicamente señalamos todas las palabras de no código como errores incorregibles. De este modo, como se muestra en (c), podemos utilizar el mismo código de distancia 4 para detectar errores hasta de 3 bits, pero corregiremos los no errores (c = 0, d = 3).

2.15.3 Códigos de Hamming

En 1950, R. W. Hamming describió un método general para construir códigos con una distancia mínima de 3, que ahora se conocen como *códigos de Hamming*. Para cualquier valor de i, su método produciría un código de $(2^i - 1)$ bits con i bits de verificación y $2^i - 1 - i$ bits de información. Los códigos de distancia 3 con un número más pequeño de bits de información se obtienen al eliminar bits de información de un código de Hamming con un número más grande de bits.

código de Hamming

Las posiciones de bits en una palabra de código de Hamming pueden numerarse desde 1 hasta $2^i - 1$. En este caso, cualquier posición cuyo número sea una potencia de 2 será un bit de verificación, y las posiciones restantes serán bits de información, como se

DECISIONES, DECISIONES Los nombres *decoding* y *decoder* tienen sentido, puesto que son solamente perturbaciones de distancia 1 de *deciding* y *decider*.

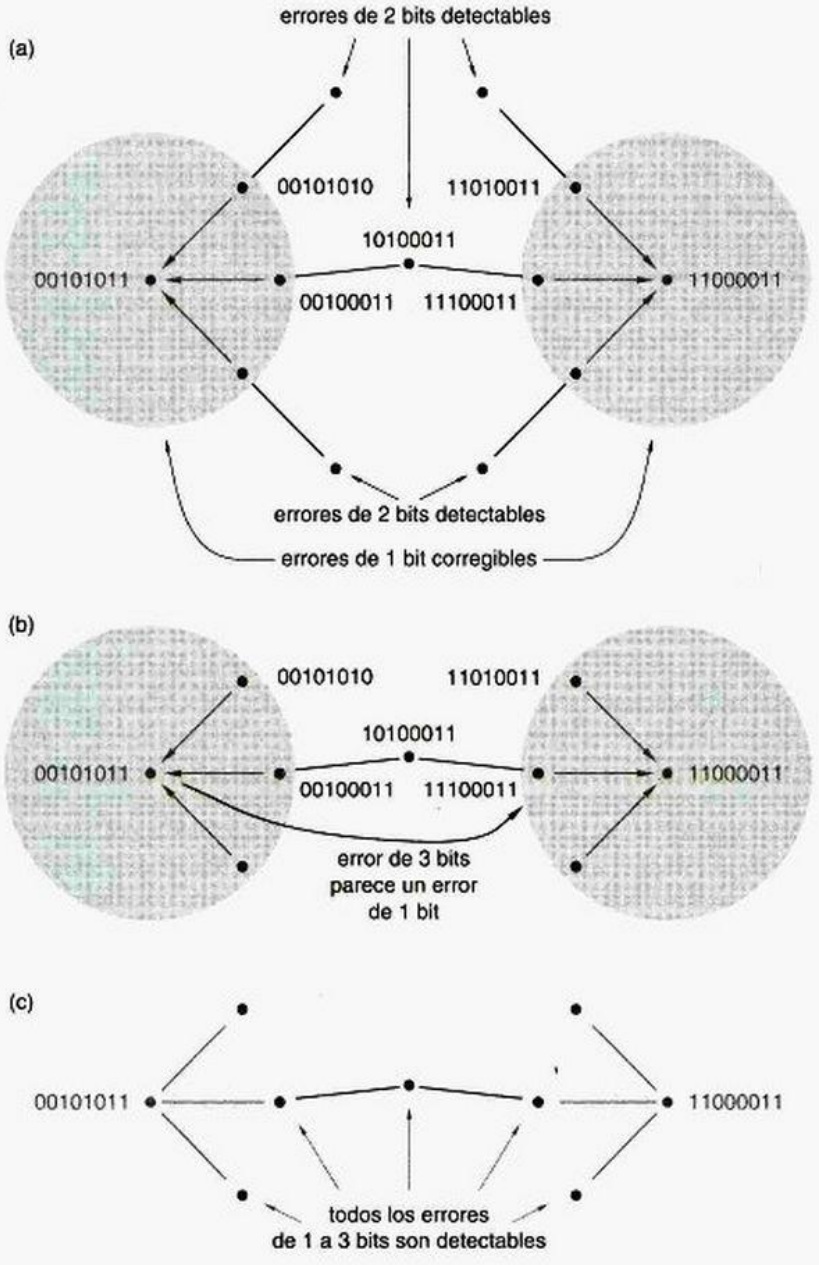


Figura 2-12
Algunas palabras de código y palabras de no código en un código de distancia 4, de 8 bits: a) corrección de errores de 1 bit y detección de errores de 2 bits; b) "corrección" incorrecta de un error de 3 bits; ausencia de corrección de errores pero detección de errores de hasta 3 bits.

matriz de verificación de paridad

especifican mediante una *matriz de verificación de paridad*. Como se muestra en la figura 2-13(a), cada bit de verificación se agrupa con las posiciones de información cuyos números tienen un 1 en el mismo bit cuando se expresan en binario. Por ejemplo, el bit de verificación 2 (010) se agrupa con los bits de información 3 (011), 6 (110) y 7 (111). Para una combinación determinada de bits de información, cada bit de verificación se elige para producir paridad par, es decir, de modo que el número total de unos en su grupo sea par.

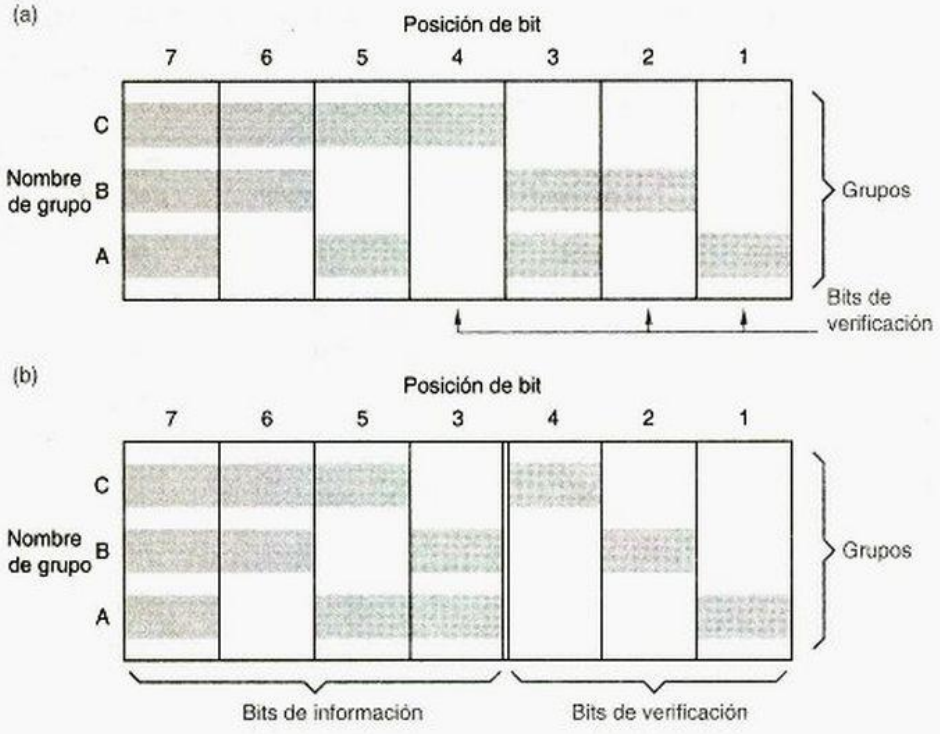


Figura 2-13
Matrices de verificación de paridad para códigos de Hamming de 7 bits: a) con posiciones de bit en orden numérico; b) con bits de verificación y bits de información por separado.

Tradicionalmente, las posiciones de los bits en una matriz de verificación de paridad y las palabras de código resultantes se reacomodan de modo que todos los bits de verificación se encuentren a la derecha, como en la figura 2-13(b). Las primeras dos columnas de la tabla 2-14 muestran las palabras de código resultantes.

Podemos probar que la distancia mínima de un código de Hamming es 3 al verificar que se debe hacer un cambio de 3 bits (como mínimo) a una palabra de código para obtener otra palabra de código. Es decir, probaremos que un cambio de 1 bit o de 2 bits en una palabra de código producirá una palabra de no código.

Si cambiamos un bit de una palabra de código, en la posición j, entonces cambiamos la paridad de cada grupo que contenga la posición j. Puesto que todo bit de información está contenido en por lo menos un grupo, al menos un grupo tiene paridad incorrecta, y el resultado es una palabra de no código.

¿Qué ocurre si cambiamos dos bits, en las posiciones j y k? Los grupos de paridad que contengan ambas posiciones j y k todavía tendrán paridad correcta, puesto que la paridad no se ve afectada cuando se modifican un número par de bits. Sin embargo, puesto que j y k son diferentes, sus representaciones binarias difieren en por lo menos un bit, correspondiendo a uno de los grupos de paridad. Este grupo tiene solamente un bit cambiado, lo que resulta en una paridad incorrecta y una palabra de no código.

Si usted comprende esta prueba, también podrá entender que las reglas de numeración de posición para construir un código de Hamming son una simple consecuencia de la prueba. Para la primera parte de la prueba (errores de 1 bit), requerimos que los números de posición sean distintos de cero. Y para la segunda parte (errores de 2 bits), necesitamos que ningún par de posiciones tengan el mismo número. De esta forma, con un número de

Tabla 2-14 Palabras de código en códigos de Hamming de distancia 3 y distancia 4 con cuatro bits de información.

Código 3 de distancia mínima		*Código 4 de distancia mínima*	
Bits de información	***Bits de paridad***	***Bits de información***	***Bits de paridad***
0000	000	0000	0000
0001	011	0001	0111
0010	101	0010	1011
0011	110	0011	1100
0100	110	0100	1101
0101	101	0101	1010
0110	011	0110	0110
0111	000	0111	0001
1000	111	1000	1110
1001	100	1001	1001
1010	010	1010	0101
1011	001	1011	0010
1100	001	1100	0011
1101	010	1101	0100
1110	100	1110	1000
1111	111	1111	1111

posición de bit i, se puede construir un código de Hamming hasta con $2^i - 1$ posiciones de bit.

decodificador de corrección de errores

La prueba sugiere también cómo podemos diseñar un *decodificador de corrección de errores* para aplicarlo a una palabra de código de Hamming que se recibe. Primero, verificamos todos los grupos de paridad; si todos tienen paridad par, entonces se asume que la palabra recibida es correcta. Si uno o más grupos tienen paridad impar, entonces se supone que ha ocurrido un error simple. El patrón de los grupos que tienen paridad impar (llamado el *síndrome*) debe coincidir con una de las columnas de la matriz de verificación de paridad; se supone que la posición correspondiente del bit contiene el valor equivocado y está complementado. Por ejemplo, usando el código definido por la figura 2-13(b), supongamos que recibimos la palabra 0101011. Los grupos B y C tienen paridad impar, correspondiente a la posición 6 de la matriz de verificación de paridad (el síndrome es 110, o 6). Al complementar el bit en la posición 6 de la palabra recibida, determinamos que la palabra correcta es 0001011.

síndrome

Un código de Hamming de distancia 3 se puede modificar fácilmente para incrementar su distancia mínima a 4. Simplemente agregamos un bit de verificación más, el cual se selecciona de modo tal que la paridad de todos los bits, incluyendo el nuevo, sea par. Como sucede en el código de paridad par de 1 bit, este bit asegura la detección de todos los errores que afectan un número impar de bits. En particular, cualquier error de 3 bits puede detectarse. Ya demostramos que los otros bits de paridad pueden detectar los errores de 1 y 2 bits, así que la distancia mínima del código modificado debe ser 4.

Los códigos de Hamming de distancia 3 y distancia 4 se utilizan comúnmente para detectar y corregir errores en los sistemas de memoria de las computadoras, especialmente en grandes computadoras "mainframes" donde los circuitos de memoria tienen que ver con la mayoría de las fallas del sistema. Estos códigos son especialmente atractivos para palabras de memoria sumamente amplias, puesto que el número requerido de bits de paridad aumenta lentamente con la amplitud de la palabra de memoria, tal como se muestra en la tabla 2-15.

Tabla 2-15 Tamaños de palabra de códigos de Hamming de distancia 3 y de distancia 4.

	Códigos 3 de distancia mínima		*Códigos 4 de distancia mínima*	
Bits de información	***Bits de paridad***	***Bits totales***	***Bits de paridad***	***Bits totales***
1	2	3	3	4
≤ 4	3	≤ 7	4	≤ 8
≤ 11	4	≤ 15	5	≤ 16
≤ 26	5	≤ 31	6	≤ 32
≤ 57	6	≤ 63	7	≤ 64
≤ 120	7	≤ 127	8	≤ 128

2.15.4 Códigos CRC

Más allá de los códigos de Hamming, se han desarrollado otros códigos de detección y corrección de errores. Los más importantes que casualmente incluyen los códigos de Hamming son los *códigos de verificación de redundancia cíclica (CRC, cyclic-redundancy-check)*. Se ha desarrollado una gran cantidad de teoría para estos códigos, que abarcan tanto sus propiedades de detección y corrección de errores como el diseño de sus codificadores y decodificadores de bajo costo (véanse Referencias).

código de verificación de redundancia cíclica (CRC)

Dos aplicaciones importantes de los códigos CRC se encuentran en las unidades controladoras de disco y en las redes de datos. En una unidad de disco, cada bloque de datos (por lo regular de 512 bytes) está protegido mediante un código CRC, de modo que los errores que están en el interior de un bloque pueden detectarse y, en algunas unidades, corregirse. En una red de datos, cada paquete de datos termina con bits de verificación en un código CRC. Los códigos CRC para ambas aplicaciones fueron seleccionados debido a sus propiedades de detección de errores ráfaga. Además de los errores de bit simple,

pueden detectar errores de múltiples bits que estén agrupados en forma contigua, en el interior del bloque (o paquete) del disco. Los errores de múltiples bits tienen una mayor probabilidad de incidencia que los errores de bits que tienen una distribución aleatoria, debido a las probables causas físicas que producen los errores en ambas aplicaciones (defectos de superficie en las unidades de disco y ráfagas de ruido en los enlaces de comunicación).

2.15.5 Códigos bidimensionales

código bidimensional

Otra manera de obtener un código que maximice la distancia mínima es construir un *código bidimensional*, como se ilustra en la figura 2-14(a). Los bits de información se acomodan conceptualmente en un arreglo bidimensional y se proporcionan bits de paridad para verificar tanto los renglones como las columnas. Un código $C_{\text{renglón}}$ con distancia mínima $d_{\text{renglón}}$ se utiliza para los renglones y un posiblemente diferente código C_{columna} con distancia mínima d_{columna} se utiliza para las columnas. Es decir, los bits de paridad de renglón se seleccionan de manera que cada renglón es una palabra de código en $C_{\text{renglón}}$ y los bits de paridad de columna se eligen de manera que es una palabra de código en C_{columna}. (Los bits de paridad de "esquina" pueden seleccionarse de acuerdo con cualquier código.) La distancia mínima del código bidimensional es el producto de $d_{\text{renglón}}$ y d_{columna}; de hecho, los códigos bidimensionales se denominan en ocasiones *códigos de producto*.

código de producto

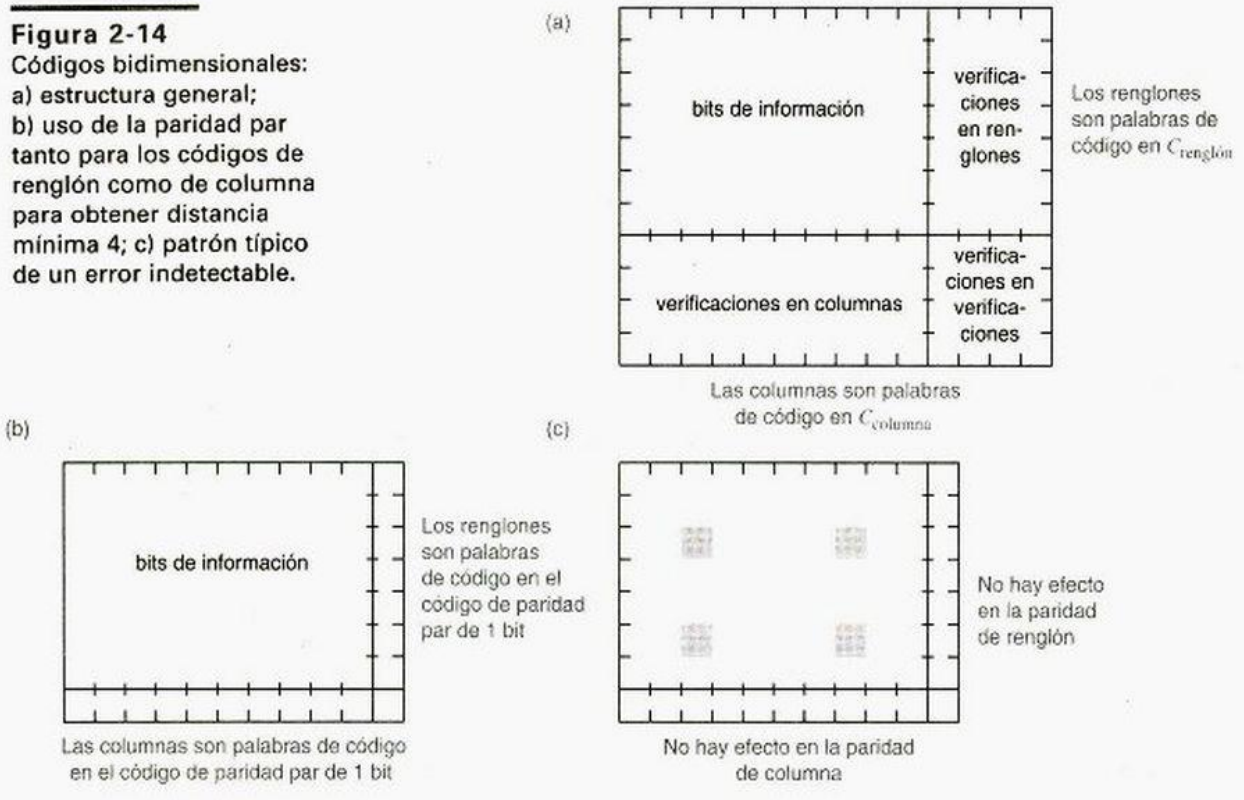


Figura 2-14
Códigos bidimensionales: a) estructura general; b) uso de la paridad par tanto para los códigos de renglón como de columna para obtener distancia mínima 4; c) patrón típico de un error indetectable.

Como se ilustra en la figura 2-14(b), el código bidimensional más simple utiliza códigos de paridad par de 1 bit para los renglones y columnas, y tiene una distancia mínima de 2 • 2, o 4. Se puede demostrar fácilmente que la distancia mínima es 4 si usted comprende que cualquier patrón de uno, dos o tres bits en error causa paridad incorrecta de un renglón, una columna o ambos. A fin de obtener un error no detectable, al menos cuatro bits deben ser cambiados en un patrón rectangular como en (c).

Los procedimientos de detección y corrección de errores para este código son directos. Supongamos que estamos leyendo información un renglón a la vez. A medida que leemos cada renglón, verificamos su código de renglón. Si se detecta un error en un renglón, no podemos decir cuál bit es erróneo de la verificación de renglón únicamente. Sin embargo, suponiendo que solamente un renglón se encuentre mal, podemos reconstruirlo formando el OR Exclusivo bit por bit de las columnas, omitiendo el renglón malo, pero incluyendo el renglón de verificación de columna.

Para obtener una distancia mínima aún más grande, puede emplearse un código de Hamming de distancia 3 o 4 para el código del renglón, de la columna o de ambos. También es posible construir un código en tres o más dimensiones, con distancia mínima igual al producto de las distancias mínimas en cada dimensión.

Una aplicación importante de los códigos bidimensionales se encuentra en sistemas de almacenamiento RAID. *RAID* son las siglas en inglés de "arreglo redundante de discos económicos". En este esquema se utilizan $n + 1$ unidades de disco idénticas para almacenar valores de datos de n discos. Por ejemplo, ocho unidades de 8 gigabytes podrían emplearse para almacenar 64 gigabytes de datos no redundantes y un noveno disco de 8 gigabytes se utilizaría para almacenar información de verificación. *RAID*

La figura 2-15 muestra el esquema general de un código bidimensional para un sistema RAID; cada unidad de disco es considerada como un renglón en el código. Cada unidad almacena m bloques de datos, donde un bloque contiene por lo regular 512 bytes. Por ejemplo, una unidad de 8 gigabytes almacenaría alrededor de 16 millones de bloques. Como se muestra en la figura, cada bloque incluye sus propios bits de verificación en un código CRC, para detectar errores dentro de ese bloque. Las primeras n unidades almacenan los datos no redundantes. Cada bloque en la unidad $n + 1$ almacena bits de

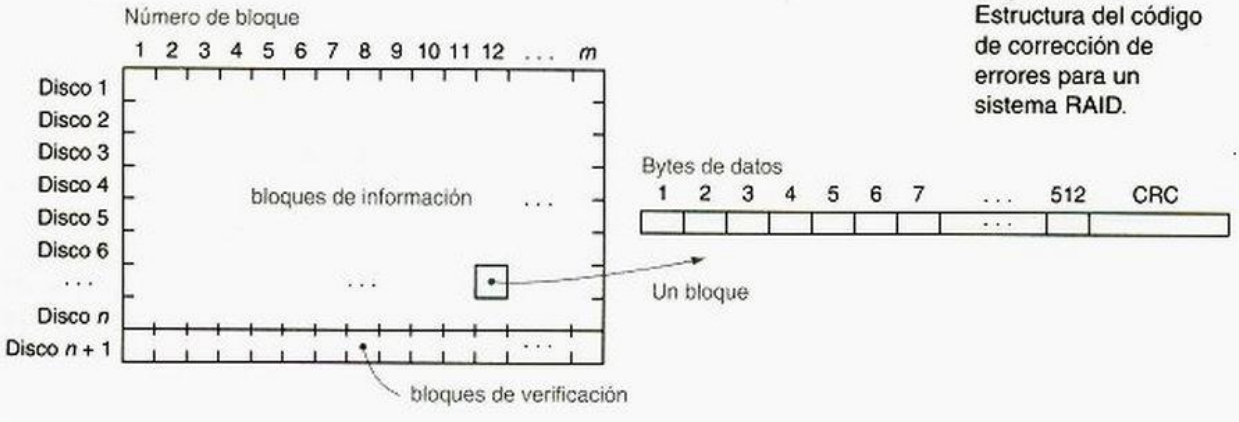


Figura 2-15 Estructura del código de corrección de errores para un sistema RAID.

paridad para los bloques correspondientes en las primeras n unidades. Es decir, cada bit i en la unidad $n + 1$, bloque b, se elige de modo que existe una cantidad par de unos en el bloque b, posición de bit i, en todas las unidades.

Una vez en operación, los errores en los bloques de información son detectados por el código CRC. Cuando se detecta un error en el bloque de una unidad de disco, se puede reconstruir el contenido correcto de ese bloque calculando la paridad de los bloques correspondientes en las demás unidades, incluyendo la unidad $n + 1$. ¡Aunque esto requiere de n operaciones adicionales de lectura en el disco, es preferible a la pérdida de sus datos! Las operaciones de escritura requieren también de accesos extras al disco, para actualizar el bloque de verificación correspondiente cuando se escribe un bloque de información (véase el ejercicio 2.46). Puesto que las escrituras en disco son mucho menos frecuentes que las lecturas en aplicaciones típicas, este gasto extra por lo regular no es un problema.

2.15.6 Códigos de suma de verificación

La operación de verificación de paridad que hemos empleado en las subsecciones anteriores es esencialmente una suma de bits módulo 2 (la suma módulo 2 de un grupo de bits es 0, si la cantidad de unos en el grupo es par, y 1, si es impar). La técnica de suma modular puede aplicarse a otras bases, aparte de la base 2, para formar dígitos de verificación.

Por ejemplo, una computadora almacena información como un conjunto de bytes de 8 bits. Se considera que cada byte puede tener un valor decimal desde 0 hasta 255. Por tanto, podemos utilizar una suma módulo 256 para verificar los bytes. Formamos un byte de verificación simple, llamado una *suma de verificación*, que es la suma módulo 256 de todos los bytes de información. El *código de suma de verificación* resultante puede detectar cualquier error de *byte* simple, puesto que un error de este tipo generará una suma recalculada de bytes cuyo resultado no coincidirá con la suma de verificación.

suma de verificación
código de suma de verificación

Los códigos de suma de verificación también pueden utilizar diferentes módulos de suma. En particular, los códigos de suma de verificación que usan la suma de complemento a unos, módulo 255, son importantes debido a sus propiedades computacionales especiales y de detección de errores; cabe indicar que se utilizan para verificar encabezados de paquetes en el ubicuo Internet Protocol (vea la sección Referencias).

código de suma de verificación de complemento a unos

2.15.7 Códigos m de n

Los códigos 1 de n y m de n que presentamos en la sección 2.13 tienen una distancia mínima de 2, puesto que cambiando solamente un bit se modifica el número total de unos en una palabra de código, y por tanto produce una palabra de no código.

Estos códigos tienen otra propiedad que se aplica en la detección de errores: pueden localizar errores múltiples unidireccionales. En un *error unidireccional*, todos los bits erróneos cambian en la misma dirección (los ceros cambian a unos, o viceversa). Esta propiedad es muy útil en sistemas donde el mecanismo del error predominante tiende a cambiar todos los bits en la misma dirección.

error unidireccional

2.16 Códigos para el almacenamiento y la transmisión de datos en serie

2.16.1 Datos en paralelo y en serie

datos en paralelo

La mayor parte de las computadoras y otros sistemas digitales transmiten y almacenan datos en un formato *paralelo*. En la transmisión de datos en paralelo, se proporciona una línea de señal independiente para cada bit de una palabra de datos. En el almacenamiento de datos en paralelo, todos los bits de una palabra de datos se pueden escribir o leer en forma simultánea.

El costo de los formatos en paralelo es elevado para algunas aplicaciones. Por ejemplo, la transmisión en paralelo de bytes de datos en la red telefónica requiere ocho líneas telefónicas y el almacenamiento paralelo de bytes de datos en un disco magnético requeriría una unidad de disco con ocho cabezas independientes de lectura/escritura. Los formatos *en serie* permiten la transmisión o el almacenamiento de datos a razón de un bit a la vez, reduciendo así el costo del sistema en muchas aplicaciones.

datos en serie

La figura 2-16 ilustra algunas de las ideas básicas en la transmisión de datos en serie. Una señal de reloj repetitiva, CLOCK en la figura, define la velocidad a la cual se transmiten los datos, un bit por cada ciclo de reloj. De este modo, la *velocidad de los bits*, en bits por segundo (bps), equivale numéricamente a la frecuencia del reloj en ciclos por segundo (hertz, o Hz).

velocidad de transmisión de bits, bps

El recíproco de la velocidad de los bits se conoce como el *tiempo de bit* y numéricamente es igual al periodo del reloj en segundos (s). Esta cantidad de tiempo está reservada en la línea de datos en serie (marcada como SERDATA en la figura) para cada bit que se transmite. El tiempo que ocupa cada bit se conoce como *celda de bit*. El formato de la señal real que aparece en la línea durante cada celda de bit depende del *código de línea*. En el código de línea más simple, denominado *No retorno a cero (NRZ, Non-Return-to-Zero)*, se transmite un 1 al colocar un 1 en la línea para toda la celda de bit, y un 0 se transmite como un 0. Los códigos de línea más complejos tienen otras reglas, como lo discutiremos en la subsección siguiente.

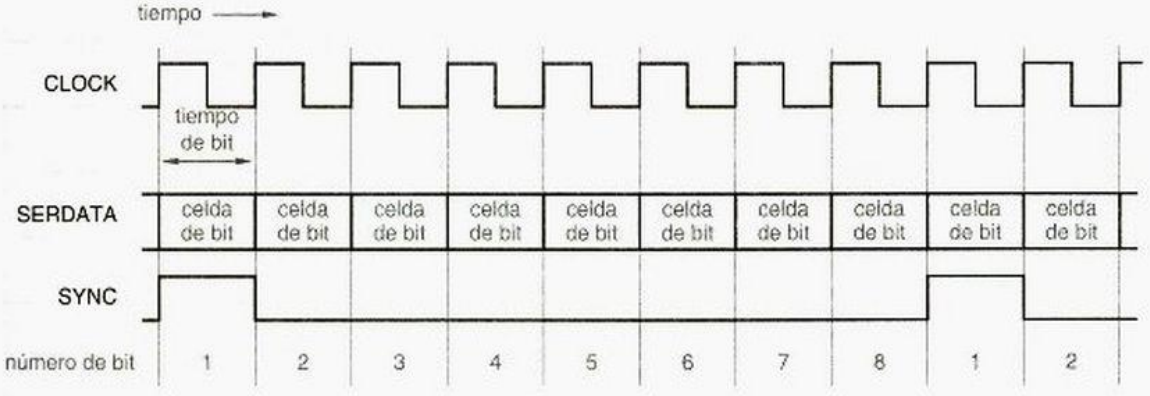
tiempo de bit

celda de bit

código de línea

No retorno a cero (NRZ)

Figura 2-16 Conceptos básicos para la transmisión de datos en serie.



señal de sincronización

Sin importar el código de línea, un sistema de almacenamiento o transmisión de datos en serie necesita algún mecanismo para identificar la importancia de cada bit en el flujo de datos en serie. Por ejemplo, supongamos que se transmiten bytes de 8 bits en serie. ¿Cómo podemos decir cuál es el primer bit de cada byte? Una *señal de sincronización*, identificada como SYNC en la figura 2-16, proporciona la información necesaria; la cual es 1 para el primer bit de cada byte.

Evidentemente, necesitamos un mínimo de tres señales para recuperar un flujo de datos en serie: un reloj para definir las celdas de bits, una señal de sincronización para definir las fronteras de la palabra y los datos en serie. En algunas aplicaciones, como en el caso de la interconexión de módulos (o tarjetas) en una computadora o sistema de telecomunicación, se utiliza un conductor independiente para cada una de estas señales, ya que reducir el número de alambres por conexión de n a tres representa un ahorro considerable. La sección 8.5.4 muestra un ejemplo de un sistema de datos en serie con tres alambres (conductores).

En muchas aplicaciones, el costo de tener tres señales independientes es bastante elevado (por ejemplo, tres líneas telefónicas, tres cabezas de lectura/escritura). Es común que estos sistemas combinen las tres señales en un solo flujo de datos en serie y por tanto, utilizan circuitos analógicos y digitales complejos para recuperar la información de reloj y sincronización del flujo de datos.

*2.16.2 Códigos de línea en serie

Los códigos de línea que se utilizan con mayor frecuencia para la transmisión de datos en serie se ilustran en la figura 2-17. En el código NRZ, cada valor de bit se envía en la línea para toda la celda de bit. Éste constituye el esquema de codificación más simple y confiable para la transmisión a poca distancia. Sin embargo, la señal de reloj debe enviarse con los datos para definir las celdas de bit. De otro modo, el receptor no podría determinar cuántos ceros o unos están representados por un nivel continuo 0 o 1. Por ejemplo, sin un reloj para definir las celdas de bit, la forma de onda NRZ en la figura 2-17 podría interpretarse erróneamente como 01010.

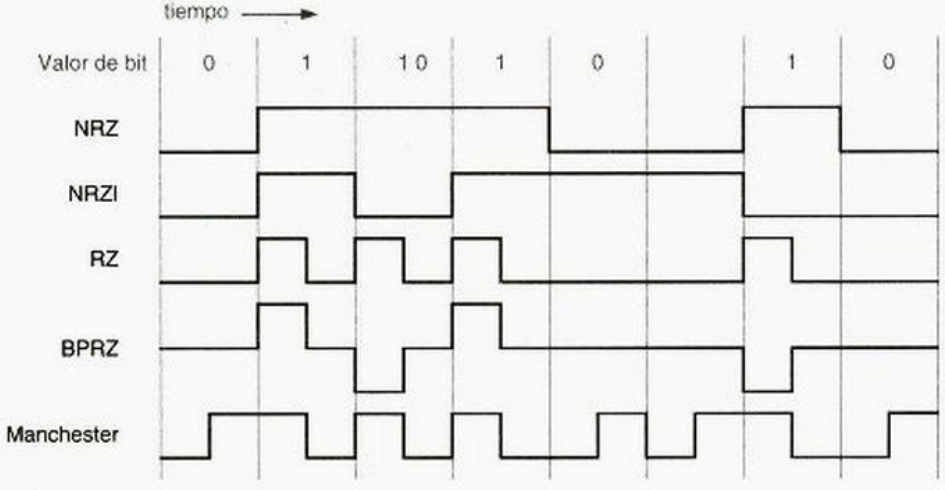


Figura 2-17
Códigos de línea usados comúnmente para datos en serie.

circuito de sincronización de fase digital (DPLL)

Un *circuito de sincronización de fase digital (DPLL, digital phase-locked loop)* es un circuito analógico/digital que se puede utilizar para recuperar una señal de reloj de un flujo de datos en serie. El DPLL funciona solamente si el flujo de datos en serie contiene suficientes transiciones de 0 a 1 y de 1 a 0 que "indican" al DPLL en qué momento se originan las transiciones originales del reloj. Con los datos codificados en NRZ, el DPLL trabaja solamente si los datos no contienen algún flujo continuo y extenso de unos o ceros.

dispositivos sensibles a la transición

Algunos dispositivos de almacenamiento y transmisión en serie son *sensibles a la transición*; no pueden transmitir o almacenar niveles absolutos 0 o 1, solamente transiciones entre dos niveles discretos. Por ejemplo, una cinta o disco magnéticos almacenarán información mediante el cambio de polaridad en la magnetización del elemento, en regiones que corresponden a los bits almacenados. Cuando se recupera la información, no es factible determinar la polaridad de magnetización absoluta de una región, solamente se detecta que la polaridad cambia entre una región y la siguiente.

Los datos que se almacenan en el formato NRZ en dispositivos sensibles a la transición no se recuperan sin cierta ambigüedad; los datos en la figura 2-17 pueden interpretarse como 01110010 o 10001101. El código de *No retorno a cero invertido en unos (NRZI, Non-Return-to-Zero Invert-on-1s)* supera esta limitación al enviar un 1 como el opuesto al nivel que se envió durante la celda de bit anterior, y un 0 como el mismo nivel. Un DPLL puede recuperar la señal de reloj de los datos codificados en NRZI siempre y cuando los datos no contengan algún flujo extenso y continuo de ceros.

No retorno a cero invertido en unos (NRZI)

Retorno a cero (RZ)

El código de Retorno a Cero (RZ) es semejante al NRZ excepto que, para un bit 1, el nivel 1 se transmite solamente por una fracción del tiempo de bit, generalmente 1/2. Con este código, los patrones de datos que contengan muchos unos generan muchas transiciones para que un DPLL pueda utilizarlas a fin de recuperar el reloj. No obstante, como sucede en otros códigos de línea, una cadena de ceros no tiene transiciones, y una larga cadena de ceros hace imposible la recuperación del reloj.

equilibrio de CD

Otro requerimiento de algunos dispositivos de transmisión, tales como enlaces de fibra óptica de alta velocidad, es que el flujo de datos en serie debe estar *equilibrado en CD*. En otras palabras, la cantidad de unos debe ser igual a la de ceros. Si en el flujo de datos existe alguna componente de CD de larga duración (debido a una mayor cantidad de unos que de ceros, o viceversa), aparecerá una polarización en el receptor que reducirá su capacidad para distinguir de manera confiable los unos y los ceros.

En general, los datos NRZ, NRZI o RZ no ofrecen ninguna garantía de equilibrio de CD; nada puede evitar que un flujo de datos tenga una larga cadena de palabras con más unos que ceros o viceversa. Sin embargo, el equilibrio de CD puede conseguirse mediante el uso de unos cuantos bits extra para codificar los datos del usuario en un *código equilibrado*. En este código cada palabra de código tiene una cantidad idéntica de unos y ceros, luego se procede a enviar las palabras de código en formato NRZ.

código equilibrado

Por ejemplo, en la sección 2.13 presentamos el código 8B10B, que codifica 8 bits de datos del usuario en 10 bits en un código en su mayor parte de 5 de 10. Cabe recordar que solamente existen 252 palabras de código 5 de 10, pero que hay otras $\binom{10}{4} = 210$ palabras de código 4 de 10 y un número igual de palabras de código 6 de 10. Naturalmente, estas palabras de código no están suficientemente equilibradas en CD. El código 8B10B resuelve este problema al asociar a cada valor de 8 bits que será codificado un *par* de palabras de código no equilibradas, una de 4 de 10 ("ligera") y otra de 6 de 10 ("pesada"). El codificador también sigue la pista de la *disparidad de tramos*, un bit de información simple

disparidad de tramos

KILO-, MEGA-, GIGA-, TERA-

Los prefijos K (kilo-), M(mega-), G(giga-) y T(tera-) quieren decir 10^3, 10^6, 10^9 y 10^{12}, respectivamente, cuando se hace referencia a bps, hertz, ohms, watts y la mayoría de las otras cantidades que se manejan en el campo de la ingeniería. Sin embargo, cuando nos referimos a dimensiones de memoria, los prefijos representan 2^{10}, 2^{20}, 2^{30} y 2^{40}. Históricamente, los prefijos fueron cooptados para este propósito porque los tamaños de memoria son normalmente potencias de 2, y 2^{10} (1024) es muy cercano a 1000.

Ahora bien, cuando alguien le ofrezca 50 kilodólares al año por su primer trabajo de ingeniería, ¡será asunto suyo negociar lo que quieren decir con el prefijo!

que indica si la última palabra de código no equilibrada que se transmitió era pesada o ligera. Cuando llega el momento de transmitir otra palabra de código no equilibrada, el codificador selecciona aquélla del par con el peso opuesto. Este sencillo truco hace disponibles 252 + 210 = 462 palabras de código para que el 8B10B codifique 8 bits de datos del usuario. Algunas de las palabras de código "extra" se utilizan para codificar de manera conveniente condiciones de ausencia de datos en la línea en serie, tales como IDLE, SYNC y ERROR. No se utilizan todas las palabras de código no equilibradas. Lo mismo sucede con algunas palabras de código balanceadas (como 0000011111), esto favorece a los pares no equilibrados que contienen más transiciones.

Retorno a Cero Bipolar (BPRZ)

Todos los códigos anteriores transmiten o almacenan solamente dos niveles de señal. El código de *Retorno a Cero Bipolar (BPRZ, Bipolar Return–to–Zero)* transmite con tres niveles de señal: +1, 0 y –1. El código es como el RZ excepto que los unos se transmiten alternativamente como +1 y –1; por esta razón, el código también se conoce como de *Inversión de Marca Alternada (AMI, Alternate Mark Inversion).*

Inversión de Marca Alternada (AMI)

La gran ventaja del BPRZ sobre el RZ es que está balanceado en CD. Esto permite enviar flujos de BPRZ a través de los dispositivos de transmisión que toleran una componente de CD, como sucede en las líneas telefónicas acopladas por transformador. De hecho, el código BPRZ se ha utilizado durante décadas en los enlaces telefónicos digitales T1, donde las señales analógicas de voz se transportan como flujos de 8000 muestras digitales de 8 bits por segundo que se transmiten en formato BPRZ a través de canales en serie de 64 Kbps.

Como sucede con el RZ, es posible recuperar una señal de reloj de flujo BPRZ siempre y cuando no existan demasiados ceros en un renglón. Aunque la TPC (Compañía Telefónica, por sus siglas en inglés) no tiene control sobre lo que usted dice (al menos, no en este momento), aún tiene una manera simple de limitar tramos de ceros. Si uno de los bytes de 8 bits (que se generan al muestrear su patrón analógico de voz) está integrado por ceros, simplemente cambian el segundo bit menos significativo a 1! Esto se conoce como *supresión del código cero* y apostaría que usted nunca lo había notado. Esto se debe a que en la mayor parte de las aplicaciones de datos de enlaces T1, usted obtiene solamente 56 Kbps de datos útiles de un canal de 64 Kbps; el LSB de cada byte siempre se fija a 1 para impedir que la supresión de código cero cambie los otros bits.

supresión del código cero

Manchester
difase

El último código en la figura 2-17 se denomina código *Manchester* o *difase*. La mayor ventaja de este código es que, sin tener en cuenta el patrón de datos que se transmiten, proporciona por lo menos una transición por celda de bit, esto facilita la recuperación de la señal del reloj. Como se muestra en la figura, un 0 se codifica como

ACERCA DE LA TPC (THE PHONE COMPANY, LA COMPAÑÍA DE TELÉFONOS)

Vea la película de James Coburn (1967), *The President's Analyst*, para analizar una visión divertida de la compañía telefónica. Con la creciente influencia de la tecnología digital y las comunicaciones inalámbricas económicas, el concepto de la conectividad universal, *personal* a la red telefónica que presenta la conclusión de la película ha llegado a ser mucho menos inverosímil.

una transición de 0 a 1 en la parte media de la celda de bit, y un 1 se codifica como una transición de 1 a 0. Pero la mayor ventaja del código Manchester también es su mayor debilidad. Puesto que tiene más transiciones por celda de bit que otros códigos, necesita un mayor ancho de banda en el dispositivo para transmitir bits a una velocidad determinada. No obstante, el ancho de banda no es un problema en los cables coaxiales que se utilizaban en las redes originales de área local (Ethernet) para transportar datos en serie codificados en código Manchester a la velocidad de 10 Mbps (megabits por segundo).

Referencias

La presentación en las primeras nueve secciones de este capítulo está basada en el capítulo 4 de la obra *Microcomputer Architecture and Programming*, de John F. Wakerly (Wiley, 1981). El lector encontrará discusiones precisas, minuciosas y entretenidas de estos temas también en *Seminumerical Algorithms*, tercera edición, de Donald E. Knuth (Addison-Wesley, 1997). A los estudiantes inclinados hacia las matemáticas les parecerán excelentes los análisis de las propiedades de la aritmética y los sistemas numéricos de Knuth, y todos los lectores disfrutarán las ideas y la historia que presenta el texto.

Las descripciones de los circuitos de lógica digital para operaciones aritméticas, así como también una introducción a las propiedades de diversos sistemas numéricos, aparecen en la obra *Computer Arithmetic*, de Kai Hwang (Wiley, 1979). El libro *Decimal Computation* de Hermann Schmid (Wiley, 1974) contiene una profunda descripción de las técnicas para la aritmética BCD.

La obra *Microcomputer Architecture and Programming: The 68000 Family*, de John F. Wakerly (Wiley, 1989) presenta una introducción a los algoritmos para la multiplicación y división binarias y a la aritmética de punto flotante. Una discusión más profunda de las técnicas aritméticas y sistemas numéricos de punto flotante puede hallarse en *Introduction to Arithmetic for Digital Systems Designers* de Shlomo Waser y Michael J. Flynn (Holt, Rinehart and Winston, 1982).

campos finitos

Los códigos CRC se basan en la teoría de los *campos finitos* que desarrolló el matemático francés Évariste Galois (1811-1832), poco antes de morir en un duelo a manos de un oponente político. La obra clásica que aborda los códigos de detección y corrección de errores es *Error-Correcting Codes* de W. W. Peterson y E. J. Weldon, Jr. (MIT Press, 1972, segunda edición); sin embargo, solamente recomiendo este libro a los lectores que dominan las matemáticas. Una introducción más accesible a la codificación se puede encontrar en la obre *Error Control Coding: Fundamentals and Applications* de S. Lin y D. J. Costello, Jr. (Prentice Hall, 1983). Otra introducción orientada a la comunicación para la teoría de la codificación se encuentra en *Error-Control Techniques for*

Digital Communication de A. M. Michelson y A. H. Levesque (Wiley-Interscience, 1985). Las aplicaciones de hardware de códigos en sistemas de cómputo se analizan en la obra *Error-Detecting Codes, Self-Checking Circuits and Applications de* John F. Wakerly (Elsevier/North-Holland, 1978).

Como se muestra en la referencia anterior del autor, los códigos de suma de verificación de complemento a uno tienen la capacidad de detectar largas ráfagas de errores unidireccionales; esto es muy útil en los canales de comunicación donde todos los errores tienden a estar en la misma dirección. Las propiedades especiales de cómputo de estos códigos permiten su aplicación en el cálculo de sumas de verificación mediante programas de software, lo anterior tiene aplicaciones importantes en el Protocolo de Internet; véase RFC-1071 y RFC-1141. Las solicitudes para comentarios (RFC, Requests for Comments) se archivan en muchos lugares de la red; solamente busque "RFC".

La obra *Introduction to Communications Engineering* de R. M. Gagliardi (Wiley-Interscience, 1988, segunda edición) presenta una introducción a las técnicas de codificación para la transmisión de datos en serie, e incluye el análisis matemático del rendimiento y los requerimientos de ancho de banda de diversos códigos. La obra *Computer Storage Systems and Technology* de Richard Matick (Wiley-Interscience, 1977) presenta una atractiva introducción a los códigos en serie que se utilizan en cintas y discos magnéticos.

La estructura del código 8B10B y la lógica que lo soporta se explica de manera agradable en la patente original de IBM de Peter Franaszek y Albert Widmer, U.S. patent number 4,486,739 (1984). Ésta y casi todas las patentes de Estados Unidos expedidas después de 1971 se encuentran en la Web, en la dirección `www.patents.ibm.com`.

Problemas propuestos

2.1 Realice las siguientes conversiones de sistemas numéricos:

(a) $1101011_2 = ?_{16}$ (b) $174003_8 = ?_2$

(c) $10110111_2 = ?_{16}$ (d) $67.24_8 = ?_2$

(e) $10100.1101_2 = ?_{16}$ (f) $F3A5_{16} = ?_2$

(g) $11011001_2 = ?_8$ (h) $AB3D_{16} = ?_2$

(i) $101111.0111_2 = ?_8$ (j) $15C.38_{16} = ?_2$

2.2 Convierta los siguientes números octales en binarios y hexadecimales:

(a) $1023_8 = ?_2 = ?_{16}$ (b) $761302_8 = ?_2 = ?_{16}$

(c) $163417_8 = ?_2 = ?_{16}$ (d) $552273_8 = ?_2 = ?_{16}$

(e) $5436.15_8 = ?_2 = ?_{16}$ (f) $13705.207_8 = ?_2 = ?_{16}$

2.3 Convierta los siguientes números hexadecimales en binarios y octales:

(a) $1023_{16} = ?_2 = ?_8$ (b) $7E6A_{16} = ?_2 = ?_8$

(c) $ABCD_{16} = ?_2 = ?_8$ (d) $C350_{16} = ?_2 = ?_8$

(e) $9E36.7A_{16} = ?_2 = ?_8$ (f) $DEAD.BEEF_{16} = ?_2 = ?_8$

2.4 ¿Cuáles son los valores octales de los cuatro bytes de 8 bits en el número de 32 bits que tiene la representación octal 12345670123_8?

2.5 Convierta los siguientes números en decimales:

(a) $1101011_2 = ?_{10}$ (b) $174003_8 = ?_{10}$

(c) $10110111_2 = ?_{10}$ (d) $67.24_8 = ?_{10}$

(e) $10100.1101_2 = ?_{10}$ (f) $F3A5_{16} = ?_{10}$

(g) $12010_3 = ?_{10}$ (h) $AB3D_{16} = ?_{10}$

(i) $7156_8 = ?_{10}$ (j) $15C.38_{16} = ?_{10}$

2.6 Realice las siguientes conversiones de sistemas numéricos:

(a) $125_{10} = ?_2$ (b) $3489_{10} = ?_8$

(c) $209_{10} = ?_2$ (d) $9714_{10} = ?_8$

(e) $132_{10} = ?_2$ (f) $23851_{10} = ?_{16}$

(g) $727_{10} = ?_5$ (h) $57190_{10} = ?_{16}$

(i) $1435_{10} = ?_8$ (j) $65113_{10} = ?_{16}$

2.7 Sume los siguientes pares de números binarios, mostrando todos los acarreos:

(a) 110101 + 11001 (b) 101110 + 100101 (c) 11011101 + 1100011 (d) 1110010 + 1101101

2.8 Repita el problema 2.7 usando la resta en vez de la suma, y mostrando los préstamos en lugar de los acarreos.

2.9 Sume los siguientes pares de números octales:

(a) 1372 + 4631 (b) 47135 + 5125 (c) 175214 + 152405 (d) 110321 + 56573

2.10 Sume los pares siguientes de números hexadecimales:

(a) 1372 + 4631 (b) 4F1A5 + B8D5 (c) F35B + 27E6 (d) 1B90F + C44E

2.11 Escriba las representaciones de complemento a unos y complemento a dos, de magnitud con signo de 8 bits, para cada uno de estos números decimales: +18, +115, +79, –49, –3, –100.

2.12 Indique si ocurre o no desbordamiento cuando se suman los siguientes números de complemento a dos de 8 bits:

(a) 11010100 + 10101011 (b) 10111001 + 11010110 (c) 01011101 + 00100001 (d) 00100110 + 01011010

2.13 ¿Cuántos errores pueden detectarse por un código con distancia mínima d?

2.14 ¿Cuál es el número mínimo de bits de paridad que se requieren para obtener un código bidimensional, de distancia 4 con n bits de información?

Ejercicios

2.15 Aquí tenemos un problema para abrir su apetito: ¿Cuál es el equivalente hexadecimal de 61453_{10}?

2.16 Cada una de las siguientes operaciones aritméticas es correcta en por lo menos un sistema numérico. Determine las posibles bases de los números en cada operación.

(a) $1234 + 5432 = 6666$ (b) $41 / 3 = 13$

(c) $33/3 = 11$ (d) $23+44+14+32 = 223$

(e) $302/20 = 12.1$ (f) $14 = 5$

2.17 La primera expedición a Marte encontró sólo las ruinas de una civilización. De los artefactos e imágenes, los exploradores dedujeron que las criaturas que produjeron esta civilización eran seres de cuatro piernas con un tentáculo que se ramificaba al final en un número de "dedos" prensiles. Después de mucho estudio, los exploradores fueron capaces de traducir las matemáticas marcianas. Encontraron que la siguiente ecuación:

$$5x^2 - 50x + 125 = 0$$

con las soluciones indicadas $x = 5$ y $x = 8$. El valor $x = 5$ parecía bastante legítimo, pero $x = 8$ requería alguna explicación. Entonces los exploradores reflexionaron en la manera en que se desarrolló el sistema numérico de la Tierra, y hallaron evidencia de que el sistema marciano tenía una historia semejante. ¿Cuántos dedos diría usted que tenían los marcianos? (De *The Bent of Tau Beta Pi*, febrero, 1956.)

2.18 Supongamos que un número B de $4n$ bits está representado por un número hexadecimal H de n dígitos. Demuestre que el complemento a dos de B está representado por el complemento a 16 de H. Establezca y demuestre una proposición similar para la representación octal.

2.19 Repita el ejercicio 2.18 usando el complemento a uno de B y el complemento a 15 de H.

2.20 Dado un entero x en el intervalo $-2^{n-1} \leq x \leq 2^{n-1} - 1$, definimos $[x]$ como la representación de complemento a dos de x, expresada como un número positivo: $[x] = x$ si $x \geq 0$ y $[x] = 2^n - |x|$ si $x < 0$, donde $|x|$ es el valor absoluto de x. Demuestre que las reglas de la suma del complemento a dos dadas en la sección 2.6 son correctas, probando que la siguiente ecuación es siempre verdadera:

$$[x + y] = ([x] + [y]) \text{ módulo } 2^n$$

(*Sugerencias:* considere cuatro casos basados en los signos de x y de y. Sin pérdida de generalidad, se puede suponer que $|x| \geq |y|$.)

2.21 Repita el ejercicio 2.20 utilizando reglas y expresiones apropiadas para la suma del complemento a unos.

2.22 Establezca una regla de desbordamiento para la suma de dos números en complemento a dos en términos de operaciones de conteo en la representación modular de la figura 2-3.

2.23 Demuestre que un número en complemento a dos puede ser convertido a una representación con más bits mediante la *extensión de signo*. Es decir, dado un número X en complemento a dos de n bits, muestre que la representación en complemento a dos de m bits de X, donde $m > n$, puede ser obtenida agregando $m - n$ copias del bit de signo de X a la izquierda de la representación de n bits de X.

2.24 Demuestre que un número de complemento a dos puede convertirse a una representación con menos bits eliminando los bits de mayor orden. Es decir, dado un número X en complemento a dos de n bits, demuestre que el número Y en complemento a dos de m bits obtenido al descartar los d bits más a la izquierda de X representa el mismo número que X, si y sólo si los todos bits descartados igualan el bit de signo de Y.

2.25 ¿Por qué es inconsistente la puntuación de "complemento a dos" y "complemento a uno"? (Véanse las primeras dos citas en la sección de Referencias.)

2.26 Un sumador binario de n bits puede ser utilizado para efectuar una operación de resta sin signo de n bits $X - Y$, realizando la operación $X + y + 1$, donde X y Y son números sin signo de n bits y la Y representa el complemento bit a bit de Y. Demuestre este hecho como sigue. Primero, pruebe que $(X - Y) = (X + Y + 1) - 2^n$. Segundo, demuestre que el acarreo de salida del sumador de n bits es lo opuesto al préstamo de la resta de n bits. Es decir, muestre que la operación $X - Y$ produce un préstamo de salida de la posición MSB si y sólo si la operación $X + Y + 1$ *no* produce un acarreo de salida de la posición MSB.

2.27 En la mayoría de los casos, el producto de dos números de complemento a dos de n bits requiere menos de $2n$ bits para representarlo. De hecho, existe solamente un caso en el cual $2n$ bits son necesarios. Encuentre cuál es este caso.

2.28 Demuestre que un número de complemento a dos puede ser multiplicado por 2 al desplazarlo una posición hacia la izquierda, con un acarreo de 0 en la posición del bit menos significativo y despreciando cualquier acarreo fuera de la posición del bit más significativo, suponiendo que no hay desbordamiento. Establezca la regla para detectar el desbordamiento.

2.29 Establezca y pruebe la exactitud de una técnica semejante a la que se describe en el ejercicio 2.28, para multiplicar un número de complemento a uno por 2.

2.30 Demuestre cómo restar números BCD, estableciendo las reglas para generar préstamos y aplicando un factor de corrección. Demuestre cómo se aplican sus reglas a cada una de las restas siguientes: 9 – 3, 5 – 7, 4 – 9, 1 – 8.

2.31 ¿Cuántas codificaciones diferentes de estado binario de 3 bits son posibles para el controlador de semáforos de la tabla 2-12?

2.32 Enumere todas las fronteras "malas" en el disco de codificación mecánica de la figura 2-5, donde una posición incorrecta puede ser detectada.

2.33 Como una función de n, ¿cuántas fronteras "malas" existen en un disco de codificación mecánica que utiliza un código binario de n bits?

2.34 Los transpondedores (emisores-receptores automáticos de identificación) de altitud a bordo en las aeronaves comerciales y privadas utilizan código Gray para codificar las lecturas de altitud que se transmiten a los controladores de tráfico aéreo. ¿Por qué?

2.35 Un foco incandescente se tensiona cada vez que se enciende, de modo que en algunas aplicaciones el tiempo de vida del foco está limitado por el número de ciclos de encendido/apagado en lugar del tiempo total que ilumina. Utilice sus conocimientos de códigos para sugerir una manera de duplicar el tiempo de vida de focos de 3 intensidades en tales aplicaciones.

2.36 Como una función de n, ¿cuántos subcubos distintos se tienen de un cubo n?

2.37 Encuentre una manera de dibujar un cubo 3 sobre una hoja de papel (u otros objetos bidimensionales) de modo que ninguna de las líneas se crucen, o demuestre que eso sea imposible.

2.38 Repita el ejercicio 2.37 para un cubo 4.

2.39 Escriba una fórmula que nos dé el número de subcubos m de un cubo n para un valor específico de m. (Su respuesta debería ser una función de n y m.)

2.40 Defina grupos de paridad para un código Hamming de distancia 3 con 11 bits de información.

2.41 Escriba las palabras de código de un código de Hamming con un bit de información.

2.42 Exponga el patrón para un error de 3 bits que no es detectado si los bits de paridad de "esquina" no se incluyen en los códigos bidimensionales de la figura 2-14.

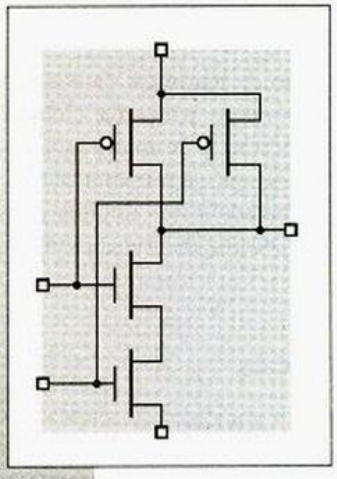
2.43 El *índice de un código* es la razón del número de bits de información con respecto al número total de bits en una palabra de código. Los índices altos, cercanos a 1, son deseables para una eficaz transmisión de la información. Construya una gráfica que compare los índices de códigos de paridad de distancia 2 y códigos Hamming de distancia 3 y 4 hasta de 100 bits de información.

2.44 ¿Qué tipo de código de distancia 4 tiene un índice mayor: un código bidimensional o un código de Hamming? Apoye su respuesta con una tabla del estilo de la tabla 2-15, incluyendo el índice así como también el número de bits de paridad y de información de cada código hasta llegar a 100 bits de información.

2.45 Muestre cómo construir un código de distancia 6 con cuatro bits de información. Escriba una lista de sus palabras de código.

2.46 Describa las operaciones que deben realizarse en un sistema RAID para escribir nuevos datos en el bloque de información b en la unidad d de manera que los datos puedan ser recuperados en el caso de un error en el bloque b en cualquier unidad. Minimice el número de accesos a disco requeridos.

2.47 Del mismo modo que en la figura 2-17, dibuje las formas de onda para el patrón de bits 10101110 cuando se envía en serie utilizando los códigos NRZ, NRZI, RZ, BPRZ y Manchester, suponiendo que los bits sean transmitidos en orden de izquierda a derecha.



capítulo 3

Circuitos digitales

A pesar de la rimbombancia publicitaria de la mercadotecnia, vivimos en un mundo analógico, no en uno digital. Los voltajes, corrientes y otras cantidades físicas en los circuitos reales toman valores que son infinitamente variables, dependiendo de las propiedades de los dispositivos reales que integran los circuitos. Debido a que los valores reales son continuamente variables, podríamos utilizar una cantidad física tal como un voltaje de señal en un circuito para representar un número real (por ejemplo, 3.14159265358979 volts representan la constante matemática *pi* con 14 dígitos decimales de precisión).

Desgraciadamente, la estabilidad y exactitud en las cantidades físicas son difíciles de obtener en los circuitos reales. Éstos se ven afectados por las tolerancias de fabricación de sus componentes, cambios en la temperatura, cambios en el voltaje de alimentación, los rayos cósmicos y ruido creado por otros circuitos, entre otras cosas. Si empleamos un voltaje analógico para representar *pi*, podemos hallar que en vez de tener una constante matemática absoluta, *pi* varía en un intervalo del 10% o más.

También, muchas operaciones matemáticas y lógicas pueden ser difíciles o imposibles de realizar con cantidades analógicas. Mientras que es posible con algo de astucia construir un circuito analógico cuyo voltaje de salida sea la raíz cuadrada de su voltaje de entrada, nunca se ha construido un circuito analógico con 100 entradas y 100 salidas, cuyas salidas sean un conjunto de voltajes idéntico al conjunto de voltajes de entrada, pero ordenados aritméticamente.

El propósito de este capítulo es proporcionar un sólido conocimiento práctico de los aspectos eléctricos de los circuitos digitales, suficiente para que usted entienda y construya sistemas y circuitos reales. Como veremos en los capítulos posteriores, mediante el uso de modernas herramientas de software es posible "construir" circuitos

de manera abstracta, utilizando lenguajes de diseño de hardware para especificar los diseños y simuladores para probar su funcionamiento. Pero, para crear circuitos reales y de calidad, ya sea a nivel de tarjeta de circuito impreso o de microcircuitos o chips, usted necesita comprender la mayor parte del material que se presenta en este capítulo. Sin embargo, si desea practicar el diseño y la simulación de circuitos abstractos en forma inmediata, puede leer únicamente la primera sección de este capítulo y postergar la lectura del resto del capítulo.

3.1 Compuertas y señales lógicas

lógica digital

La *lógica digital* oculta las dificultades del mundo analógico al mapear el conjunto infinito de valores reales de una cantidad física en dos subconjuntos que corresponden solamente a dos números posibles o *valores lógicos*: 0 y 1. Como resultado, los circuitos lógicos digitales pueden ser analizados y diseñados funcionalmente, haciendo uso del álgebra de conmutación, tablas y otros medios abstractos para describir la operación de los ceros y unos en un circuito.

valores lógicos

dígito binario
bit

Un valor lógico, 0 o 1, se denomina frecuentemente *dígito binario*, o *bit*. Si una aplicación requiere más de dos valores discretos, pueden utilizarse bits adicionales, con un conjunto de n bits que representan 2^n valores diferentes.

En la tabla 3-1 se pueden apreciar ejemplos de algunos fenómenos físicos que se utilizan para representar bits en ciertas tecnologías digitales modernas (y no tan modernas). En la mayor parte de estos fenómenos, existe una región indefinida entre los estados 0 y 1 (por ejemplo, voltaje = 1.8 V, luz disminuida, capacitor débilmente cargado, etc.). Esta región indefinida es necesaria para que los estados 0 y 1 puedan definirse sin ambigüedad y se detecten en forma confiable. Las señales de ruido pueden alterar con mayor facilidad los resultados si las fronteras que separan los estados 0 y 1 están muy cerca una de la otra.

BAJO (LOW)
ALTO (HIGH)

Cuando se analizan los circuitos lógicos electrónicos tales como CMOS y TTL, los diseñadores digitales con frecuencia utilizan las palabras "BAJO" ("LOW") y "ALTO" ("HIGH") en lugar de "0" y "1" para recordar que están tratando con circuitos reales, no con cantidades abstractas:

BAJO	Señal que está comprendida en el intervalo de voltajes algebraicamente más bajos; se interpreta como un 0 lógico.
ALTO	Señal que está comprendida en el intervalo de voltajes algebraicamente más altos; se interpreta como un 1 lógico.

lógica positiva

lógica negativa

Nótese que las asignaciones de 0 y 1 a BAJO y ALTO son algo arbitrarias. La asignación de 0 a BAJO y de 1 a ALTO parece más natural y se denomina *lógica positiva*. La asignación opuesta, de 1 a BAJO y 0 a ALTO, no se utiliza con frecuencia y se conoce como *lógica negativa*.

amplificadores buffer

Ya que una amplia gama de valores físicos representan el mismo valor binario, la lógica digital es altamente inmune a las variaciones en el valor de los componentes, del voltaje de alimentación y al ruido. Además, se pueden utilizar circuitos que utilizan amplificadores buffer, para regenerar los valores "débiles" en "fuertes", de modo que las señales digitales puedan transmitirse en distancias arbitrarias sin que existan pérdidas de información. Por ejemplo, un buffer para lógica CMOS convierte cualquier voltaje

Tabla 3-1 Estados físicos representando bits en diferentes lógicas de computadora. y tecnologías de memoria.

	Bit de representación de estado	
Tecnología	***0***	***1***
Lógica neumática	Fluido a baja presión	Fluido a alta presión
Lógica de relevador	Circuito abierto	Circuito cerrado
Lógica metal-óxido semiconductor complementaria (CMOS)	0–1.5 V	3.5–5.0 V
Lógica de transistor-transistor (TTL)	0–0.8 V	2.0–5.0 V
Fibra óptica	Luz apagada	Luz encendida
Memoria dinámica	Capacitor descargado	Capacitor cargado
Memoria borrable, no volátil	Electrones atrapados	Electrones liberados
Memoria sólo de lectura bipolar	Fusible fundido	Fusible intacto
Memoria de burbuja	Ausencia de burbuja magnética	Presencia de burbuja magnética
Disco o cinta magnéticos	Dirección del flujo "norte"	Dirección del flujo "sur"
Memoria polimérica	Molécula en el estado A	Molécula en el estado B
Disco compacto sólo de lectura	Ausencia de pozo (orificio en la superficie)	Pozo
Disco compacto reescribible	Establecido en estado cristalino	Establecido en estado no cristalino

de entrada ALTO en una salida muy cercana a 5.0 V, y cualquier voltaje de entrada BAJO en una salida muy cercana a 0.0 V.

Se puede representar a un circuito lógico con una mínima cantidad de detalle, sencillamente en forma de una "caja negra" con un cierto número de entradas y salidas. Por ejemplo, la figura 3-1 muestra un circuito lógico con tres entradas y una salida. Sin embargo, esta representación no describe cómo responde el circuito a las señales de entrada.

Desde el punto de vista del diseño de circuito electrónico, se necesita una gran cantidad de información para describir el comportamiento eléctrico de un circuito en forma precisa. No obstante, puesto que las entradas de un circuito lógico digital pueden verse como si tomaran únicamente los valores discretos 0 y 1, la operación "lógica" del circuito puede describirse mediante una tabla en la que se ignora el comportamiento eléctrico y solamente se indican los valores discretos 0 y 1.

Figura 3-1
Representación de "caja negra" de un circuito lógico con tres entradas y una salida.

circuito combinacional
tabla de verdad

Un circuito lógico cuyas salidas dependen solamente de sus entradas actuales se denomina *circuito combinacional*. Su funcionamiento se describe completamente en una *tabla de verdad* que enumera todas las combinaciones de valores de entrada así como los valores de salida que producen los valores de entrada. La tabla 3-2 corresponde a la tabla de verdad de un circuito lógico con tres entradas X, Y y Z, y una sola salida F.

Tabla 3-2
Tabla de verdad para un circuito lógico combinacional.

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

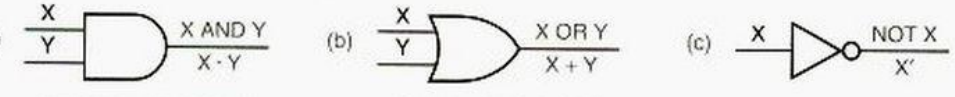
circuito secuencial
tabla de estado

Un circuito con memoria cuyas salidas dependan de la entrada actual y de la secuencia de las entradas anteriores, se conoce como *circuito secuencial*. El comportamiento de un circuito de esta clase puede describirse mediante una *tabla de estado* que especifica su salida y el estado siguiente como funciones de su entrada y estado actual. Los circuitos secuenciales se presentarán en el capítulo 7.

Como veremos más adelante en la sección 4.1, solamente se necesitan las tres funciones lógicas básicas, AND, OR y NOT para construir cualquier circuito lógico digital combinacional. La figura 3-2 ilustra las tablas de verdad y los símbolos para las "compuertas" lógicas que realizan estas funciones. Los símbolos y las tablas de verdad de las compuertas AND y OR pueden extenderse a compuertas que tengan cualquier cantidad de entradas. Las funciones de las compuertas se definen fácilmente con palabras:

compuerta AND
compuerta OR
compuerta NOT
inversor

- Una *compuerta AND* produce una salida 1 si y sólo si todas sus entradas son 1.
- Una *compuerta OR* produce una salida 1 si y sólo si una o más de sus entradas son 1.
- Una *compuerta NOT*, por lo regular conocida como un *inversor*, produce un valor de salida que es el opuesto al de su valor de entrada.

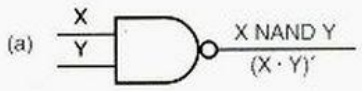


X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

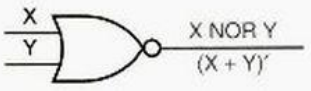
X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

Figura 3-2
Elementos lógicos básicos: a) AND; b) OR; c) NOT (inversor).



X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0



X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

Figura 3-3
Compuertas inversoras: a) NAND; b) NOR.

El círculo que está en la salida del símbolo de inversor se conoce como *burbuja de inversión* y se utiliza en éste y otros símbolos de compuerta para denotar un comportamiento "inversor". *burbuja de inversión*

Nótese que en las definiciones de las funciones AND y OR solamente tuvimos que establecer las condiciones de entrada para las cuales la salida es 1, porque sólo existe una posibilidad cuando la salida no es 1: debe ser 0.

Dos funciones lógicas más se obtienen al combinar NOT con una función AND u OR en una compuerta simple. La figura 3-3 muestra las tablas de verdad y los símbolos para estas compuertas; sus funciones también son fácilmente describibles con palabras:

- Una *compuerta NAND* produce el opuesto de una salida de compuerta AND, un 0 si y sólo si todas sus entradas son 1. *compuerta NAND*
- Una *compuerta NOR* produce el opuesto de una salida de compuerta OR, un 0 si y sólo si una o más de sus entradas son 1. *compuerta NOR*

Como sucede en las compuertas AND y OR, los símbolos y tablas de verdad para NAND y NOR pueden hacerse extensivos a compuertas que tienen cualquier cantidad de entradas.

La figura 3-4 muestra un circuito lógico que utiliza compuertas AND, OR y NOT que funcionan de acuerdo con la tabla de verdad de la tabla 3-2. En el capítulo 4 aprenderemos cómo pasar de una tabla de verdad a un circuito lógico, y viceversa, y también aprenderemos acerca de la notación de álgebra de conmutación utilizada en las figuras 3-2 hasta la 3-4.

Los circuitos lógicos reales también funcionan en otra dimensión analógica: el tiempo. Por ejemplo, la figura 3-5 es un *diagrama de temporización* que muestra cómo responde el circuito de la figura 3-4 a un patrón de señales de entrada que varían en el tiempo. El diagrama de temporización indica que las señales lógicas no cambian entre 0 y 1 de manera instantánea, y también que existe un retraso entre un cambio en la entrada *diagrama de temporización*

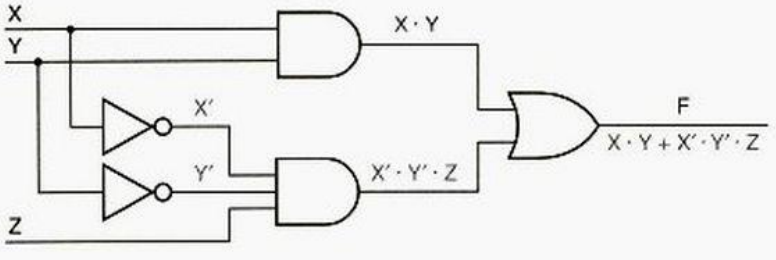


Figura 3-4
Circuito lógico con la tabla de verdad de la tabla 3-2.

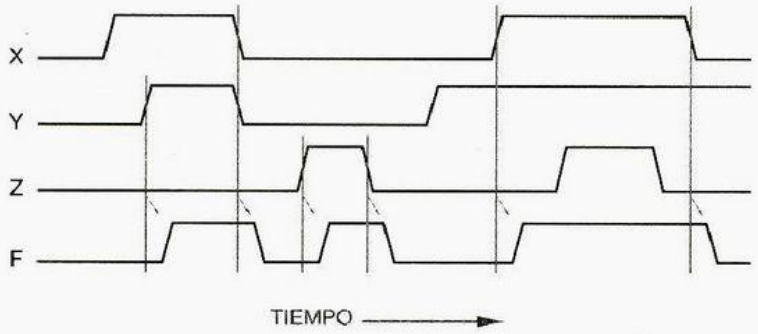


Figura 3-5
Diagrama de temporización para un circuito lógico.

y el correspondiente cambio en la salida. Más adelante en este capítulo aprenderemos algunas de las razones para este comportamiento en el tiempo y cómo se especifica y maneja en los circuitos reales. Y una vez más, aprenderemos en un capítulo posterior por qué en la mayoría de los casos se puede ignorar este comportamiento analógico en el tiempo en los circuitos secuenciales, y en cambio el circuito puede visualizarse como si se moviera entre estados discretos a intervalos precisos, definidos por una señal de reloj.

De este modo, incluso si usted no sabe nada acerca de electrónica analógica, debería ser capaz de comprender el comportamiento lógico de los circuitos digitales. Sin embargo, llega un momento en el diseño y depuración cuando todo diseñador de lógica digital debe descartar en forma temporal "la abstracción digital", y considerar el fenómeno analógico que limita o trastorna el desempeño digital. El resto de este capítulo lo preparará para ese día mediante el estudio de las características eléctricas de los circuitos lógicos digitales.

HAY ESPERANZA PARA LOS QUE NO SON INGENIEROS ELÉCTRICOS

Si a usted le molestan todas estas "cosas" eléctricas, no se preocupe, al menos por ahora. El resto de este libro está escrito para ser tan independiente de estas cuestiones tanto como sea posible. Pero usted las necesitará más adelante, si es que tiene que diseñar y construir sistemas digitales en la vida real.

3.2 Familias lógicas

Existen muchas, muchas maneras para diseñar un circuito lógico electrónico. Los primeros circuitos lógicos controlados eléctricamente, desarrollados en los Laboratorios Bell en la década de 1930, estaban basados en relevadores. A mediados de la década de 1940, la primera computadora digital electrónica, la ENIAC, utilizaba circuitos lógicos basados en tubos de vacío. La ENIAC tenía aproximadamente 18,000 tubos y una cantidad semejante de compuertas lógicas, no muchas para los estándares actuales de CIs para microprocesadores (que tienen decenas de millones de transistores). Sin embargo, la ENIAC podría lastimarlo mucho más que un chip si cayera sobre usted: ¡tenía 100 pies de longitud, 10 pies de altura, 3 pies de profundidad y consumía 140,000 watts de potencia!

diodo semiconductor
transistor bipolar de unión

La invención del *diodo semiconductor* y del *transistor bipolar de unión* permitió el desarrollo de computadoras más pequeñas, más rápidas y de mayor capacidad a fina-

les de la década de 1950. En la década 1960, la invención del *circuito integrado (CI)* permitió la fabricación de muchos diodos, transistores y otros componentes en un solo chip, y las computadoras se hicieron todavía mejores.

circuito integrado (CI)

La década de 1960 fue testigo de la introducción de las primeras familias lógicas de circuitos integrados. Una *familia lógica* es una colección de diferentes chips de circuitos integrados que tienen características similares en sus entradas, salidas y circuitería interna, pero que realizan diferentes funciones lógicas. Los chips de la misma familia pueden interconectarse para que realicen cualquier función lógica deseada. Por otra parte, los chips de familias diferentes pueden no ser compatibles; pueden utilizar diferentes voltajes de alimentación o pueden emplear diferentes condiciones de entrada y salida para representar a los valores lógicos.

familia lógica

La *familia lógica bipolar* con mayor éxito (una basada en transistores bipolares de unión) es la de *lógica transistor-transistor (TTL, transistor-transistor logic)*. Presentada por vez primera en la década de 1960, la TTL es en realidad una familia de familias lógicas que son compatibles con cada una de las otras pero que difieren en velocidad, consumo de energía y costo. Los sistemas digitales pueden mezclar componentes de varias familias TTL diferentes, de acuerdo con los objetivos de diseño y las restricciones en diferentes partes del sistema. Aunque la familia TTL fue reemplazada en gran parte por los circuitos CMOS en la década de 1990, es probable que encontremos componentes TTL en los laboratorios; por esto, presentamos las familias TTL en la sección 3.10.

familia lógica bipolar
lógica transistor-transistor (TTL)

Diez años *antes* que fuera inventado el transistor bipolar de unión, ya se habían patentado los principios de operación para otro tipo de transistor, llamado *transistor de efecto de campo metal-óxido-semiconductor (MOSFET Metal-Oxide Semiconductor Field-Effect Transistor)*, o sencillamente *transistor MOS*. Sin embargo, los transistores MOS eran difíciles de fabricar en sus primeros días, y no fue sino hasta la década de 1960 que una ola de desarrollos hizo prácticos los circuitos de memoria y lógica basados en dispositivos MOS. Aun entonces, los circuitos MOS no superaban a los circuitos bipolares en lo que respecta a la velocidad, y solamente eran atractivos en ciertas aplicaciones debido a su bajo consumo de energía y mayor nivel de integración.

transistor de efecto de campo metal-óxido semiconductor (MOSFET)

transistor MOS

A mediados de la década de 1980 los avances en el diseño de los circuitos MOS, en particular los circuitos *MOS complementario (CMOS, complementary MOS)*, provocaron un aumento súbito en su desempeño y popularidad. Con mucho, la mayor parte de los modernos circuitos integrados a gran escala, tales como microprocesadores y memorias, utilizan dispositivos CMOS. Del mismo modo, para las aplicaciones que van de pequeña a mediana escala (donde hasta hace algún tiempo, los dispositivos TTL fueron la única familia lógica a escoger), se plantea el uso de dispositivos CMOS que tienen una funcionalidad equivalente, pero que ofrecen mayor velocidad y menor consumo de energía. En la actualidad, los circuitos CMOS integran la inmensa mayoría del mercado mundial de CI.

MOS complementario (CMOS)

La lógica CMOS es la tecnología de lógica digital comercial con mayor capacidad y la más fácil de comprender. Al inicio de la siguiente sección, describiremos la estructura básica de los circuitos lógicos CMOS y presentaremos las familias lógicas CMOS que se utilizan con mayor frecuencia.

A consecuencia de la transición TTL-CMOS, muchas familias CMOS se diseñaron para que fueran un tanto compatibles con los dispositivos TTL. En la sección 3.12

mostraremos cómo se pueden integrar las familias TTL y CMOS en un sistema sencillo.

3.3 Lógica CMOS

El comportamiento funcional de un circuito lógico CMOS es muy fácil de comprender, incluso si sus conocimientos de electrónica analógica no son particularmente profundos. Los únicos bloques básicos de construcción en los circuitos lógicos CMOS son transistores MOS, los cuales describiremos en forma breve. Antes de presentar los transistores MOS y los circuitos lógicos CMOS, debemos analizar los niveles lógicos.

3.3.1 Niveles lógicos CMOS

Los elementos lógicos abstractos procesan dígitos binarios, 0 y 1. Sin embargo, los circuitos lógicos reales procesan señales eléctricas tales como niveles de voltaje. En cualquier circuito lógico existe un intervalo de voltajes (u otras condiciones del circuito) que se interpreta como un 0 lógico, y otro intervalo (que no se traslapa con el anterior) que se interpreta como un 1 lógico.

Un circuito lógico CMOS típico funciona a partir de una fuente de alimentación de 5 volts. Un circuito de esta clase interpretará cualquier voltaje que esté en el intervalo de 0 a 1.5 V como un 0 lógico, de igual forma, interpretará a cualquier voltaje que esté en el intervalo de 3.5 a 5.0 V como un 1 lógico. De este modo, las definiciones de BAJO y ALTO para la lógica CMOS de 5 V son como se ilustra en la figura 3-6. Los voltajes que están en el intervalo intermedio (1.5 – 3.5 V) solamente aparecerán durante las transiciones de señal, por tanto producirán valores lógicos indefinidos (es decir, un circuito puede interpretarlos ya sea como 0 o como 1). Los circuitos CMOS que utilizan otros voltajes de alimentación, tales como 3.3 o 2.7 volts, dividen el intervalo de voltaje en forma similar.

3.3.2 Transistores MOS

El modelo de un transistor MOS corresponde a un dispositivo de 3 terminales que actúa como una resistencia controlada por voltaje. Como lo sugiere la figura 3-7, el voltaje de entrada que se aplica en una terminal controla la resistencia entre las dos terminales restantes. En aplicaciones de lógica digital, un transistor MOS opera de modo que su resistencia siempre sea muy elevada —el transistor se encuentra "apagado" ("off")— o muy baja —el transistor se encuentra "encendido" ("on")—.

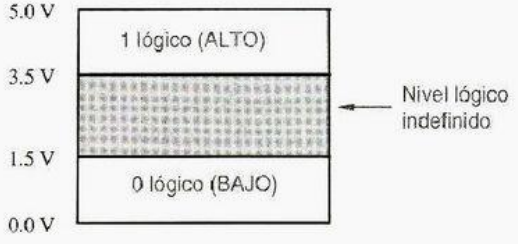


Figura 3-6
Niveles lógicos para circuitos lógicos CMOS típicos.

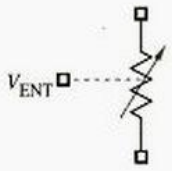


Figura 3-7
El transistor MOS como una resistencia controlada por voltaje.

Existen dos tipos de transistores MOS, de canal-*n* y de canal-*p*; los nombres se refieren al tipo de material semiconductor que se utiliza en las terminales cuya resistencia está controlada. La figura 3-8 muestra el símbolo esquemático de un *transistor MOS de canal-n (NMOS, n-channel MOS)*. Las terminales se denominan *compuerta (gate), fuente (source)* y *drenaje (drain)*. (Nótese que la "compuerta" de un transistor MOS no tiene nada que ver con una "compuerta lógica".) Como podrá observar, a partir de la orientación del símbolo del circuito, el drenaje se encuentra normalmente a un voltaje más alto que la fuente.

transistor MOS de canal-n (NMOS)
compuerta
fuente
drenaje

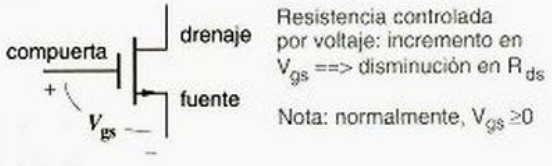


Figura 3-8
Símbolo de circuito para un transistor MOS de canal-*n* (NMOS).

El voltaje de la compuerta a la fuente (V_{gs}) en un transistor NMOS es normalmente cero o positivo. Si $V_{gs} = 0$, entonces la resistencia del drenaje a la fuente (R_{ds}) es muy alta, al menos un megaohm (10^6 ohms) o más. A medida que incrementamos V_{gs} (es decir, aumentamos el voltaje en la compuerta), R_{ds} disminuye hasta llegar a un valor muy bajo, de 10 ohms o menos en algunos dispositivos.

La figura 3-9 muestra el símbolo de circuito para un *transistor MOS de canal-p (PMOS)*. Su funcionamiento es análogo al de un transistor NMOS, excepto porque la fuente se encuentra normalmente a un voltaje mayor que el drenaje, y por lo general V_{gs} es cero o negativo. Si V_{gs} es cero, entonces la resistencia de fuente a drenaje (R_{ds}) es muy elevada. Conforme disminuimos algebraicamente V_{gs} (es decir, *disminuimos* el voltaje en la compuerta), R_{ds} disminuye hasta llegar a un valor muy bajo.

transistor MOS de canal-p (PMOS)

La compuerta de un transistor MOS tiene una impedancia muy elevada. Es decir, la compuerta está separada de la fuente y el drenaje por un material aislante que tiene una resistencia muy elevada. Sin embargo, el voltaje de compuerta crea un campo eléctrico que refuerza o retarda el flujo de corriente entre las terminales de fuente y drenaje. Éste es el "efecto de campo" en el término "MOSFET".

Sin importar el voltaje de compuerta, casi no fluye corriente de la compuerta a la fuente, o de la compuerta al drenaje para este asunto. La resistencia entre la compuerta

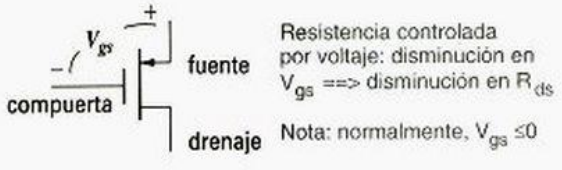


Figura 3-9
Símbolo de circuito para un transistor MOS de canal-*p* (PMOS).

IMPEDANCIA VERSUS RESISTENCIA Técnicamente, existe una diferencia entre las palabras "impedancia" y "resistencia", pero los ingenieros eléctricos con frecuencia utilizan los términos de forma intercambiable. Así lo haremos en este texto.

y las otras terminales del dispositivo es extremadamente alta, superior a un megohm. La diminuta cantidad de corriente que fluye a través de esta resistencia es muy pequeña, por lo regular es menor a un microampere (μA, 10^{-6} A), y se conoce como *corriente de fuga*.

corriente de fuga

El mismo símbolo del transistor MOS nos recuerda que no existe conexión entre la compuerta y las otras dos terminales del dispositivo. Sin embargo, la compuerta de un transistor MOS está acoplada capacitivamente a la fuente y al drenaje, como podría sugerirlo el propio símbolo. En circuitos de alta velocidad, la energía que se necesita para cargar y descargar esta capacitancia en cada transición de la señal de entrada representa una parte no trivial del consumo de energía de un circuito.

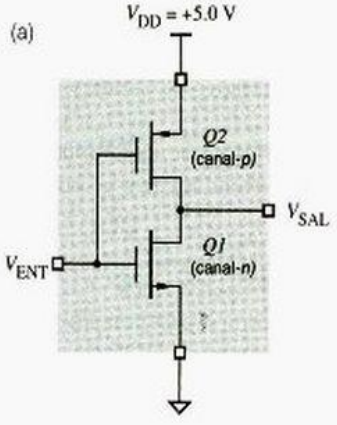
3.3.3 Circuito del inversor básico CMOS

lógica CMOS

Los transistores NMOS y PMOS se utilizan en forma complementaria para formar la *lógica CMOS*. El circuito CMOS más sencillo, un inversor lógico, requiere solamente uno de cada tipo de transistor, conectados como se muestra en la figura 3-10(a). Normalmente, el voltaje de alimentación, V_{DD} se encuentra en el intervalo de 2 a 6 V y con mucha frecuencia se establece en 5.0 V para tener compatibilidad con los circuitos TTL.

En términos ideales, el comportamiento funcional del circuito inversor CMOS puede caracterizarse mediante dos casos que se tabulan en la figura 3-10(b):

1. V_{ENT} es 0.0 V. En este caso, el transistor inferior de canal-*n*, *Q1*, está apagado, puesto que su V_{gs} es 0, pero el transistor superior de canal-*p*, *Q2*, está encendido, puesto que su V_{gs} es un valor negativo grande (–5.0 V). Por lo tanto, *Q2*, presenta solamente una pequeña resistencia entre la terminal de alimentación (V_{DD}, +5.0 V) y la terminal de salida (V_{SAL}), y el voltaje de salida es 5.0 V.



(b)

V_{ENT}	Q1	Q2	V_{SAL}
0.0 (L)	apagado	encendido	5.0 (H)
5.0 (H)	encendido	apagado	0.0 (L)

(c)

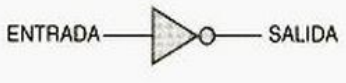


Figura 3-10
Inversor CMOS:
a) diagrama de circuito;
b) comportamiento funcional;
c) símbolo lógico.

2. V_{ENT} es 5.0 V. Aquí, *Q1* se encuentra encendido, puesto que su V_{gs} es un valor positivo grande (+5.0 V), pero *Q2* se encuentra apagado, puesto que su V_{gs} es 0. De este modo, *Q1* presenta una pequeña resistencia entre la terminal de salida y tierra, y el voltaje de salida es 0 V.

Con el comportamiento funcional anterior, el circuito se comporta claramente como un inversor lógico, puesto que una entrada de 0 V produce una salida de 5 volts, y viceversa.

Otra manera de visualizar el funcionamiento del CMOS es el uso de interruptores. Como se ilustra en la figura 3-11(a), el modelo del transistor de canal-*n* (abajo) corresponde a un interruptor normalmente abierto, y el transistor de canal-*p* (arriba) corresponde a un interruptor normalmente cerrado. La aplicación de un voltaje ALTO cambia cada interruptor al caso opuesto de su estado normal, como se muestra en (b).

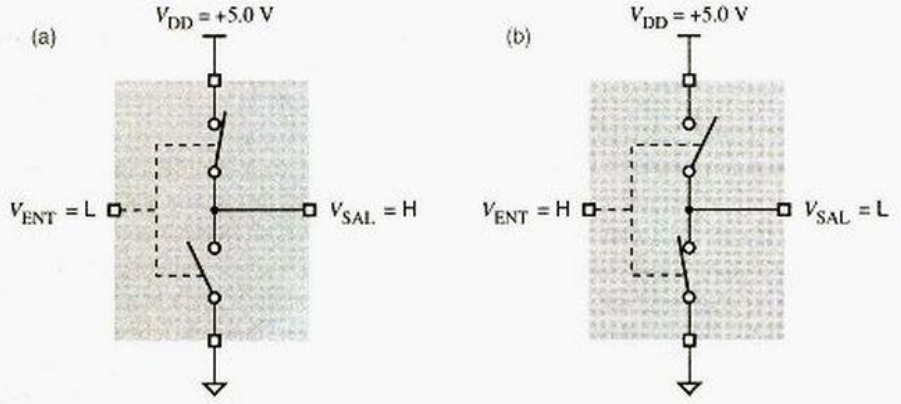


Figura 3-11
Modelo de interruptor para el inversor CMOS: a) entrada BAJO; b) entrada ALTO.

El modelo del interruptor nos lleva a una manera de dibujar circuitos CMOS que hace que su comportamiento lógico sea más evidente. Como se muestra en la figura 3-12, se emplean símbolos diferentes para los transistores de canal-*n* y de canal-*p*, a fin de reflejar su comportamiento lógico. El transistor de canal-*n* (*Q1*) se encuentra "activado", y la corriente fluye entre la fuente y el drenaje, cuando se aplica un voltaje ALTO a su compuerta; esto parece bastante natural. El transistor de canal-*p* (*Q2*) tiene el comportamiento

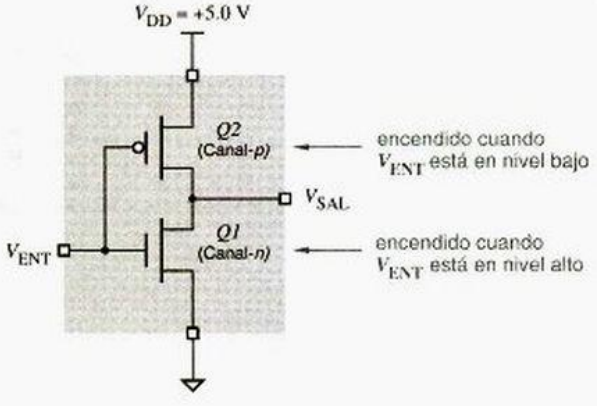


Figura 3-12
Operación lógica del inversor CMOS.

¿QUÉ TENEMOS EN UN NOMBRE?

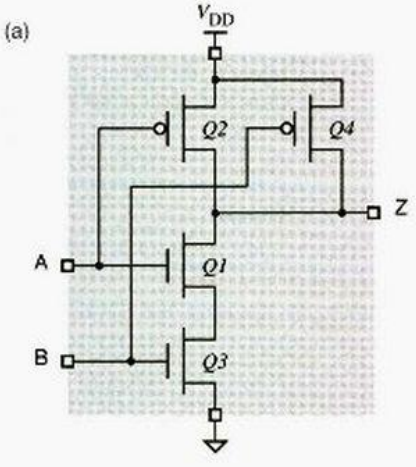
La expresión "DD" en el nombre "V_{DD}" se refiere a las terminales de *drenaje* de los transistores MOS. Esto puede parecer extraño, puesto que en el inversor CMOS V_{DD} se encuentra conectado en realidad a la terminal *fuente* de un transistor PMOS. Sin embargo, los circuitos lógicos CMOS evolucionaron a partir de los circuitos lógicos NMOS, donde la alimentación *estaba* conectada al drenaje de un transistor NMOS a través de un resistor de carga, y el nombre "V_{DD}" se quedó. También observe que en ocasiones se hace referencia a la tierra como "V_{SS}" en los circuitos CMOS y NMOS. Algunos autores y muchos fabricantes de circuitos utilizan "V_{CC}" como el símbolo para el voltaje de alimentación CMOS, puesto que este nombre se usaba en circuitos TTL, los que precedieron históricamente a CMOS. Para conseguir que usted utilice ambos, comenzaremos empleando "V_{CC}" en la sección 3.4.

opuesto. Se encuentra "activado" cuando se aplica un voltaje BAJO; la burbuja de inversión en su compuerta indica este comportamiento de inversión.

3.3.4 Compuertas CMOS NAND y NOR

Tanto la compuerta NAND como la NOR pueden construirse utilizando dispositivos CMOS. Una compuerta de k entradas utiliza k transistores de canal-p y k transistores de canal-n. La figura 3-13 muestra una compuerta NAND CMOS de 2 entradas. Si cualquiera de las entradas es BAJO, la salida Z tiene una conexión de baja impedancia a V_{DD} a través del correspondiente transistor de canal-p "encendido", y la trayectoria a tierra se bloquea mediante el correspondiente transistor de canal-n "apagado". Si ambas entradas son ALTO, la trayectoria a V_{DD} es bloqueada, y Z tiene una conexión de baja impedancia a tierra. La figura 3-14 muestra el modelo de interruptor para el funcionamiento de la compuerta NAND.

La figura 3-15 muestra una compuerta CMOS NOR. Si ambas entradas son BAJO, entonces la salida Z tiene una conexión de baja impedancia a V_{DD} a través de los transistores de canal-p "que están encendidos" y la trayectoria a tierra se encuentra bloqueada por los transistores de canal-n "que están apagados". Si cualquier entrada es ALTO, la trayectoria a V_{DD} estará bloqueada, y Z tendrá una conexión de baja impedancia a tierra.



(b)

A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	H
H	L	on	off	off	on	H
H	H	on	off	on	off	L

Figura 3-13
Compuerta NAND de 2 entradas CMOS: a) diagrama de circuito; b) tabla de función; c) símbolo lógico.

NAND VERSUS NOR Las compuertas CMOS NAND y NOR no tienen un rendimiento idéntico. Para un área de silicio dada, un transistor de canal-*n* tiene menor resistencia de "encendido" que un transistor de canal-*p*. Por lo tanto, cuando los transistores se colocan en serie, *k* transistores de canal-*n* tienen menor resistencia de "encendido" que la que tienen unos *k* transistores de canal-*p*. Como resultado, una compuerta NAND de *k* entradas es generalmente más rápida (y preferida) que una compuerta NOR de *k* entradas.

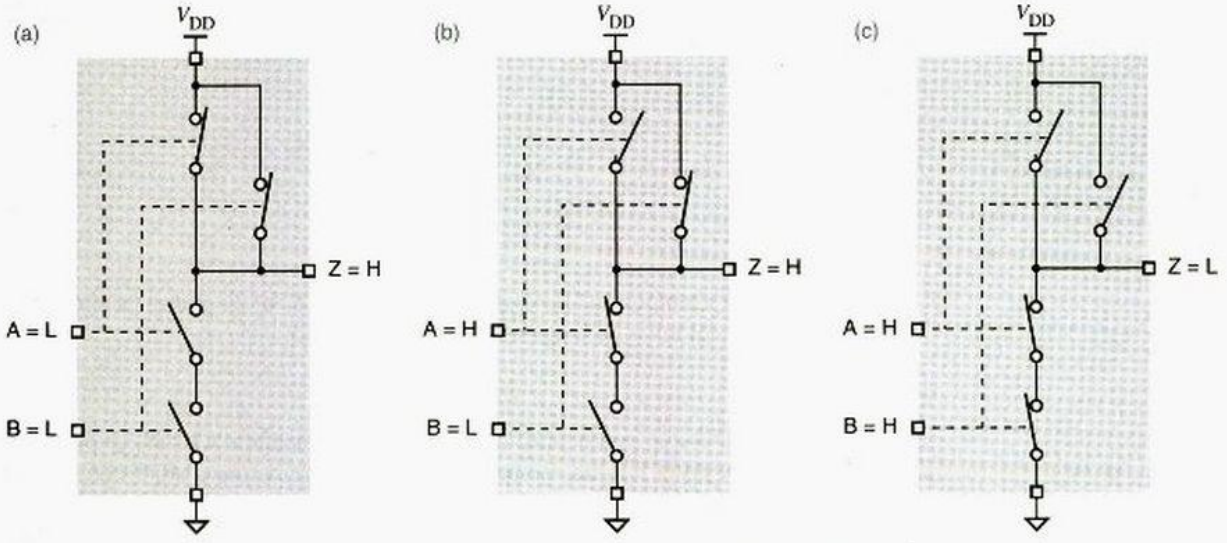
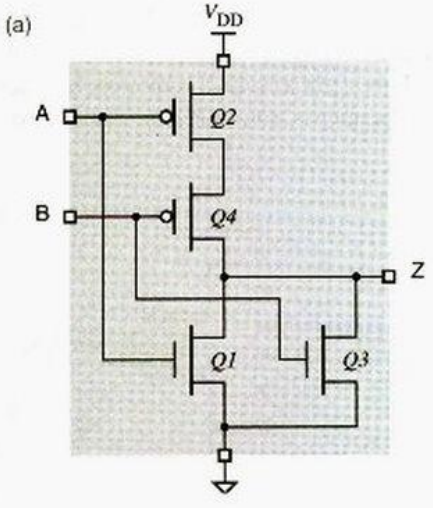


Figura 3-14 Modelo de interruptores para una compuerta NAND de 2 entradas CMOS: a) ambas entradas en BAJO; b) una entrada en ALTO; c) ambas entradas en ALTO.



(b)

A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L

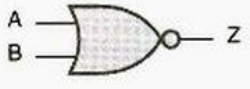
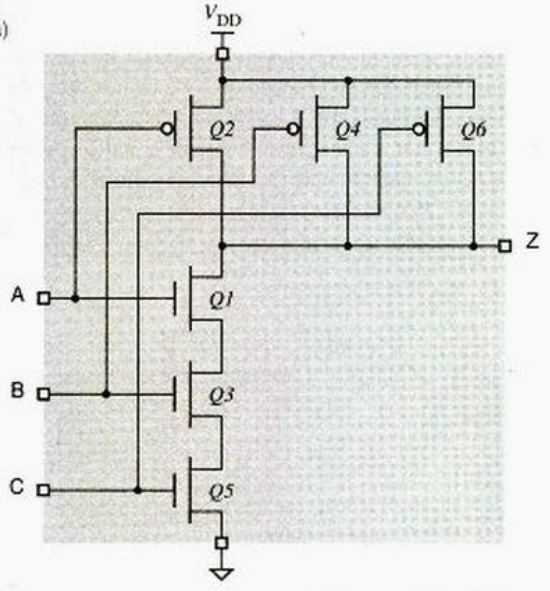


Figura 3-15 Compuerta NOR de 2 entradas CMOS: a) diagrama de circuito; b) tabla de funciones; c) símbolo lógico.

(a)



(b)

A	B	C	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	L	off	on	off	on	off	on	H
L	L	H	off	on	off	on	on	off	H
L	H	L	off	on	on	off	off	on	H
L	H	H	off	on	on	off	on	off	H
H	L	L	on	off	off	on	off	on	H
H	L	H	on	off	off	on	on	off	H
H	H	L	on	off	on	off	off	on	H
H	H	H	on	off	on	off	on	off	L

(c)

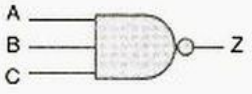


Figura 3-16 Compuerta NAND de 3 entradas CMOS: a) diagrama de circuito; b) tabla de funciones; c) símbolo lógico.

3.3.5 Fan in

Fan in

La cantidad de entradas que puede tener una compuerta en una familia lógica en particular se denomina *Fan in* de la familia. Se pueden obtener compuertas CMOS con más de dos entradas mediante la extensión de los diseños serie-paralelo de las figuras 3-13 y 3-15. Por ejemplo, la figura 3-16 muestra una compuerta CMOS NAND de 3 entradas.

En principio, se podría diseñar una compuerta CMOS NAND o NOR con una gran cantidad de entradas. Sin embargo, en la práctica, la resistencia adicional de "encendido" de los transistores en serie limita el Fan in de las compuertas CMOS, cuyo valor típico es 4 para las compuertas NOR y 6 para las compuertas NAND.

Conforme aumenta la cantidad de entradas, los diseñadores de circuitos de compuertas CMOS deben compensar este hecho incrementando el tamaño de los transistores en serie para reducir su resistencia y el correspondiente tiempo de retardo en la conmutación. No obstante, en algún punto esto llega a ser poco eficaz o poco práctico. Se pueden fabricar compuertas cada vez más rápidas y pequeñas con una gran cantidad de

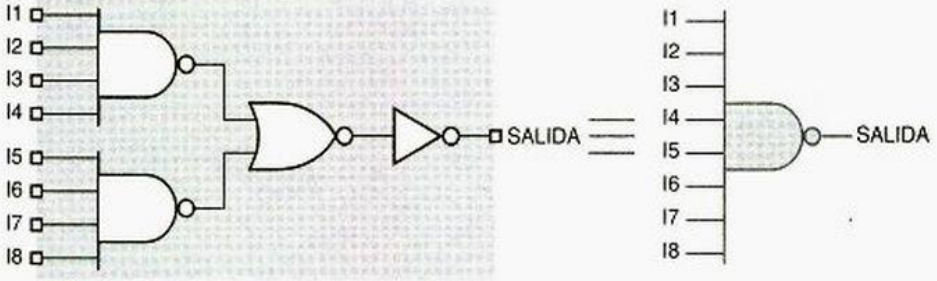
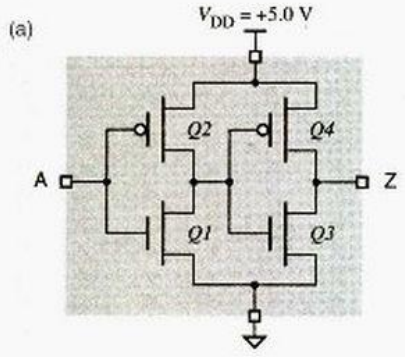


Figura 3-17 Diagrama lógico equivalente a la estructura interna de una compuerta NAND CMOS de 8 entradas.



(b)

A	Q1	Q2	Q3	Q4	Z
L	off	on	on	off	L
H	on	off	off	on	H

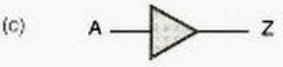


Figura 3-18
Buffer no inversor CMOS: a) diagrama de circuito; b) tabla de funciones; c) símbolo lógico.

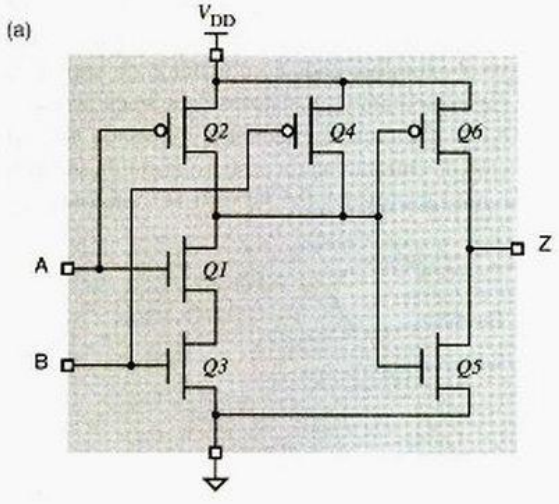
entradas si se conectan en cascada compuertas con pocas entradas. Por ejemplo, la figura 3-17 ilustra la estructura lógica de una compuerta CMOS NAND de 8 entradas. El retardo total (típico) a través de una compuerta NAND de 4 entradas, una NOR de 2 entradas y un inversor es menor que el retardo de un circuito NAND de un nivel con 8 entradas.

3.3.6 Compuertas no inversoras

En las familias lógicas CMOS, y en la mayoría de las otras familias lógicas, las compuertas más sencillas son los inversores, y las que le siguen en simplicidad son las compuertas NAND y NOR. La inversión lógica "ya está incluida" y por lo regular es imposible diseñar una compuerta no inversora con una cantidad inferior de transistores que una compuerta inversora.

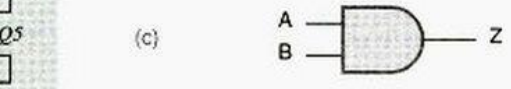
Los buffers no inversores CMOS y las compuertas AND y OR se obtienen al conectar un inversor a la salida de la correspondiente compuerta de inversión. De esta manera, la figura 3-18 muestra un buffer no inversor y la figura 3-19 ilustra una compuerta AND. Al combinar la figura 3-15(a) con un inversor se obtiene una compuerta OR.

Figura 3-19 Compuerta AND CMOS de 2 entradas: a) diagrama de circuito; b) tabla de funciones; c) símbolo lógico.

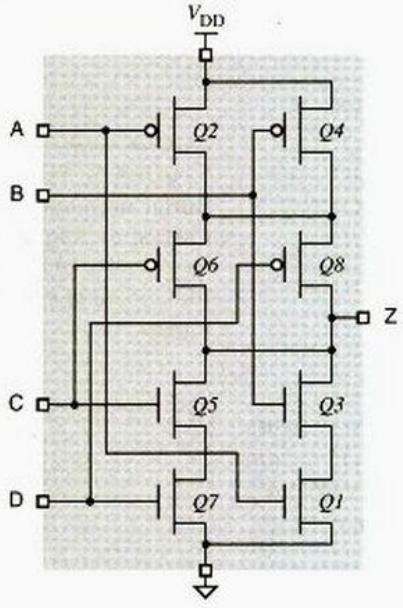


(b)

A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	off	on	off	on	on	off	L
L	H	off	on	on	off	on	off	L
H	L	on	off	off	on	on	off	L
H	H	on	off	on	off	off	on	H



(a)



(b)

A	B	C	D	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	L
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	H
L	H	H	L	off	on	on	off	on	off	off	on	H
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	H
H	L	H	L	on	off	off	on	on	off	off	on	H
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	L
H	H	L	H	on	off	on	off	off	on	on	off	L
H	H	H	L	on	off	on	off	on	off	off	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

Figura 3-20 Compuerta AND-OR-INVERSORA CMOS: a) diagrama de circuito; b) tabla de funciones.

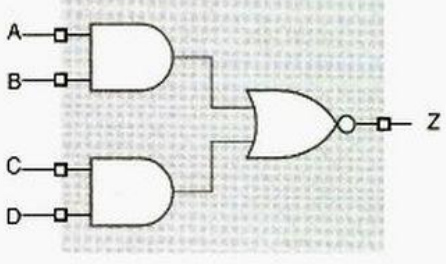
3.3.7 Compuertas CMOS AND-OR-INVERSORA y OR-AND-INVERSORA

compuerta AND-OR-INVERSORA (AOI)

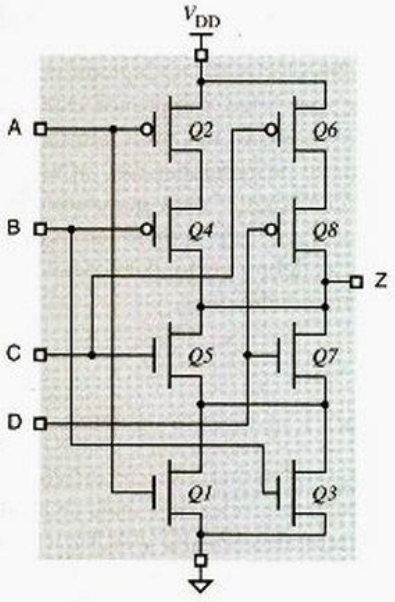
Los circuitos CMOS pueden realizar dos niveles de lógica con simplemente un solo "nivel" de transistores.*1* Por ejemplo, el circuito de la figura 3-20(a) es una *compuerta AND-OR-INVERSORA (AO)* de dos niveles y dos entradas, CMOS. La tabla de funciones para este circuito se muestra en (b) y la figura 3-21 muestra el diagrama lógico para estas funciones (el cual utiliza compuertas AND y NOR). Pueden agregarse o eliminarse transistores en este circuito para obtener una función AOI con una cantidad diferente de compuertas AND o un número diferente de entradas por compuerta AND.

El contenido de cada una de las columnas *Q1-Q8* en la figura 3-20(b) depende solamente de la señal de entrada que se conecta en la compuerta correspondiente del transistor. La última columna se construye al examinar cada combinación de entrada y determinar si Z está conectada a V_{DD} o a tierra mediante transistores "que se encienden" con esta combinación de entrada. Nótese que Z nunca se encuentra conectada *tanto a* V_{DD} *como a* tierra para ninguna combinación de entrada; en un caso así la salida sería un valor no lógico que estaría en algún lugar entre los niveles BAJO y ALTO. En este caso

Figura 3-21
Diagrama lógico para la compuerta AND-OR-INVERSORA.



(a)



(b)

A	B	C	D	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	H
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	L
L	H	H	L	off	on	on	off	on	off	off	on	L
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	L
H	L	H	L	on	off	off	on	on	off	off	on	L
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	H
H	H	L	H	on	off	on	off	off	on	on	off	L
H	H	H	L	on	off	on	off	on	off	off	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

Figura 3-22 Compuerta OR-AND-INVERSORA CMOS: a) diagrama de circuito; b) tabla de funciones.

la estructura de salida consumiría un exceso de energía debido a la conexión de baja impedancia entre V_{DD} y tierra.

También se puede diseñar un circuito que realice la función OR-AND-INVERSORA. Por ejemplo, la figura 3-22(a) muestra una *compuerta OR-AND-INVERSORA (OAI)* de dos niveles y dos entradas, CMOS. La tabla de funciones para este circuito se muestra en (b); los valores en cada columna se determinan como en el caso de la compuerta CMOS tipo AOI. La figura 3-23 muestra el diagrama lógico para la función OAI que utiliza compuertas OR y NAND.

compuerta OR-AND-INVERSORA (AOI)

La velocidad y otras características eléctricas de una compuerta CMOS tipo AOI u OAI son bastante comparables a las de una sola *compuerta* CMOS tipo NAND o NOR. Como resultado, estas compuertas son muy atractivas porque pueden realizar dos niveles de lógica (AND-OR u OR-AND) con únicamente un nivel de retardo. La mayoría de los diseñadores digitales no se preocupan por usar compuertas AOI en sus diseños discretos. Sin embargo, los dispositivos CMOS VLSI utilizan con frecuencia estas compuertas de manera interna, ya que muchas herramientas de síntesis HDL pueden convertir automáticamente la lógica AND/OR en compuertas AOI cuando es apropiado.

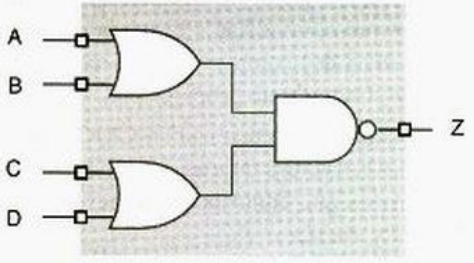


Figura 3-23
Diagrama lógico para la compuerta OR-AND-INVERSORA CMOS.

3.4 Comportamiento eléctrico de los circuitos CMOS

En las secciones 3.5 a 3.7 analizaremos los aspectos eléctricos, no lógicos, del funcionamiento de los circuitos CMOS. Es importante que usted comprenda este material antes de iniciar el diseño de circuitos reales con circuitos CMOS u otras familias lógicas. La mayor parte de este material tiene el propósito de proporcionar una infraestructura que le permita comprobar la validez de la "abstracción digital" para un circuito determinado. En particular, un diseñador de circuitos o sistemas debe proporcionar los márgenes adecuados de diseño de ingeniería a todas las etapas del proyecto, para asegurar que el circuito funcione correctamente, aun bajo las peores condiciones.

márgenes de diseño de ingeniería

3.4.1 Resumen

Los temas que discutiremos en las secciones 3.5 a 3.7 incluirán lo siguiente:

- *Niveles lógicos de voltaje.* El fabricante garantiza que los dispositivos CMOS que operan bajo condiciones normales producirán niveles de voltaje de salida que estarán comprendidos en intervalos bien definidos (BAJO y ALTO). A su vez los dispositivos reconocerán los niveles de voltaje de entrada BAJO y ALTO de algunos intervalos más amplios. Los fabricantes de dispositivos CMOS especifican estos intervalos y condiciones de operación con mucho cuidado para asegurar la compatibilidad entre diferentes dispositivos de la misma familia y para suministrar cierto grado de interconectividad (si fuera necesario) entre dispositivos de familias diferentes.
- *Márgenes de ruido de CD.* Los márgenes no negativos de ruido de CD aseguran que el voltaje BAJO de mayor nivel que produce una salida siempre será menor que el voltaje más alto que una entrada puede interpretar confiablemente como BAJO; y que el voltaje ALTO de menor nivel que produce una salida siempre será mayor que el voltaje más bajo que una entrada puede interpretar confiablemente como ALTO. Una buena comprensión de los márgenes de ruido es especialmente importante en los circuitos que emplean dispositivos de diferentes familias.
- *Fan out.* Se refiere al número y tipo de entradas que se conectan a una salida determinada. Si se conectan demasiadas entradas a una salida, los márgenes de ruido de CD del circuito pueden ser inadecuados. La cargabilidad de salida también puede afectar la velocidad con la que cambia la salida de un estado a otro.
- *Velocidad.* El tiempo que le toma a una salida CMOS cambiar del estado BAJO al estado ALTO, o viceversa, depende tanto de la estructura interna del dispositivo como de las características de los otros dispositivos que maneja, inclusive, las pistas de la tarjeta de circuito impreso que se conectan a la salida pueden afectar la velocidad. Posteriormente examinaremos dos componentes independientes de la "velocidad": el tiempo de transición y el retardo de propagación.
- *Consumo de energía.* La energía que consume un dispositivo CMOS depende de varios factores, que incluyen no sólo su estructura interna, sino también las señales de entrada que recibe, los dispositivos que controla y con qué frecuencia su salida cambia entre los estados BAJO y ALTO.

- *Ruido.* La razón principal por la que se deben proporcionar los márgenes de diseño de ingeniería es asegurar el funcionamiento adecuado del circuito en la presencia de ruido. El ruido puede ser generado por diversas fuentes; a continuación se indican varias de ellas, se indica primero la menos probable hasta llegar (en forma sorprendente) a la más probable:
  - Rayos cósmicos.
  - Campos magnéticos que provienen de maquinaria cercana.
  - Perturbaciones de la fuente de alimentación.
  - La acción de conmutación de los propios circuitos lógicos.
- *Descargas electrostáticas.* ¿Sabe usted que puede destruir un dispositivo CMOS sólo con tocarlo?
- *Salidas de drenaje abierto.* Algunas salidas de CMOS omiten los transistores habituales canal-*p* (pull-up). En el estado ALTO, una salida de este tipo se comporta esencialmente como una "no conexión", lo cual es útil en algunas aplicaciones.
- *Salidas de tres estados.* Algunos dispositivos CMOS tienen una entrada adicional de control para "habilitar la salida" que se puede utilizar para deshabilitar tanto los transistores de canal-*p* como los transistores de canal-*n* (pull-down). Se pueden conectar entre sí varias salidas de dispositivos de esta clase para crear un bus de fuente múltiple, siempre y cuando la lógica de control se arregle para que solamente se encuentre habilitada una salida a la vez.

3.4.2 Hojas de datos y especificaciones

hojas de datos

Los fabricantes de los dispositivos del mundo real proporcionan *hojas de datos* que especifican las características eléctricas y de lógica del dispositivo. La tabla 3-3 muestra la sección de especificaciones eléctricas de una hoja de datos de un dispositivo sencillo CMOS, la compuerta NAND cuádruple 54/74HC00. En general, los fabricantes pueden especificar parámetros adicionales, e incluso puede variar la forma en la que especifican los parámetros "estándar" de la tabla. Por consiguiente, también se indican los circuitos de prueba y las formas de onda que se utilizan para definir los diversos parámetros, por ejemplo como se muestran en la figura 3-24. Nótese que esta figura contiene información de algunos parámetros, además de los que se aplican al 54/74HC00.

Es probable que en este momento la mayor parte de los términos de la hoja de datos y las formas de onda de la figura no tengan sentido alguno para usted. Sin embargo, después de leer las secciones 4, 5 y 6 usted conocerá lo suficiente acerca de las características eléctricas de los circuitos CMOS y podrá comprender los aspectos más importantes de ésta o cualquier otra hoja de datos. Como diseñador lógico, usted necesitará este conocimiento para crear sistemas y circuitos confiables y robustos en el mundo real.

NO TENGA MIEDO Los estudiantes de ciencias de la computación y otros lectores que no son del área de ingeniería eléctrica no deberían temer al material de las siguientes tres secciones. Solamente se requiere de un entendimiento básico de electrónica, al nivel de la ley de Ohm.

Tabla 3-3 Hoja de especificaciones del fabricante para un dispositivo típico C-MOS, un 54/74HC00, compuerta cuádruple NAND.

CARACTERÍSTICAS ELÉCTRICAS DE CD SOBRE EL INTERVALO DE OPERACIÓN
Las siguientes condiciones se aplican a menos que se especifique lo contrario:
Comercial: $T_A = -40°C$ hasta $+85°C$, $V_{CC} = 5.0V \pm 5\%$; Militar: $T_A = -55°C$ hasta $+125°C$, $V_{CC} = 5.0V \pm 10\%$

Símbolo	Parámetro	Condiciones de prueba[1]		Mínimo	Típico[2]	Máximo	Unidad
V_{IH}	Nivel de entrada ALTO	Nivel lógico ALTO garantizado		3.15	—	—	V
V_{IL}	Nivel de entrada BAJO	Nivel lógico BAJO garantizado		—	—	1.35	V
I_{IH}	Corriente de entrada ALTO	V_{CC} = Máximo $V_I = V_{CC}$		—	—	1	μA
I_{IL}	Corriente de entrada BAJO	V_{CC} Máximo $V_I = 0$ V		—	—	−1	μA
V_{IK}	Voltaje de diodo encendido	V_{CC} = Mínimo $I_N = -18$ mA		—	−0.7	−1.2	V
I_{IOS}	Corriente de corto circuito	V_{CC} = Máximo[3] V_O = GND		—	—	−35	mA
V_{OH}	Voltaje de salida ALTO	V_{CC} = Mínimo, $V_{IN} = V_{IL}$	$I_{OH} = -20$ μA	4.4	4.499	—	V
			$I_{OH} = -4$ mA	3.84	4.3	—	V
V_{OL}	Voltaje de salida BAJO	V_{CC} = Mínimo, $V_{IN} = V_{IH}$	$I_{OL} = 20$ μA	—	.001	0.1	V
			$I_{OL} = 4$ mA		0.17	0.33	
I_{CC}	Corriente de alimentación en reposo	V_{CC} = Máximo, V_{IN} = GND o V_{CC}, $I_O = 0$		—	2	10	μA

CARACTERÍSTICAS DE CONMUTACIÓN SOBRE EL INTERVALO DE OPERACIÓN, $C_L = 50$ pF

Símbolo	Parámetro[4]	Condiciones de prueba		Mínimo	Típico	Máximo	Unidad
t_{PD}	Retardo de propagación	A o B o Y		—	9	19	ns
C_I	Capacitancia de entrada	$V_{IN} = 0V$		—	3	10	pF
C_{pd}	Capacitancia de disipación de energía por compuerta		Sin carga	—	22	—	pF

NOTAS:

1. *Para condiciones mostradas como Máximo o Mínimo, utilice el valor apropiado especificado bajo Características Eléctricas.*
2. *Los valores típicos son para $V_{CC} = 5.0$ V, +25°C temperatura ambiente.*
3. *No más de una salida debería ponerse en corto a la vez. La duración de la prueba de corto circuito no debería exceder un segundo.*
4. *Este parámetro está garantizado pero no probado*

¿QUÉ SIGNIFICA EL NÚMERO? Dos prefijos diferentes, "74" y "54", se utilizan en los números de parte de los dispositivos CMOS y TTL. Estos prefijos simplemente hacen la distinción entre las versiones comerciales y militares. Un 74HC00 es la parte comercial mientras que la 54HC00 es la versión militar.

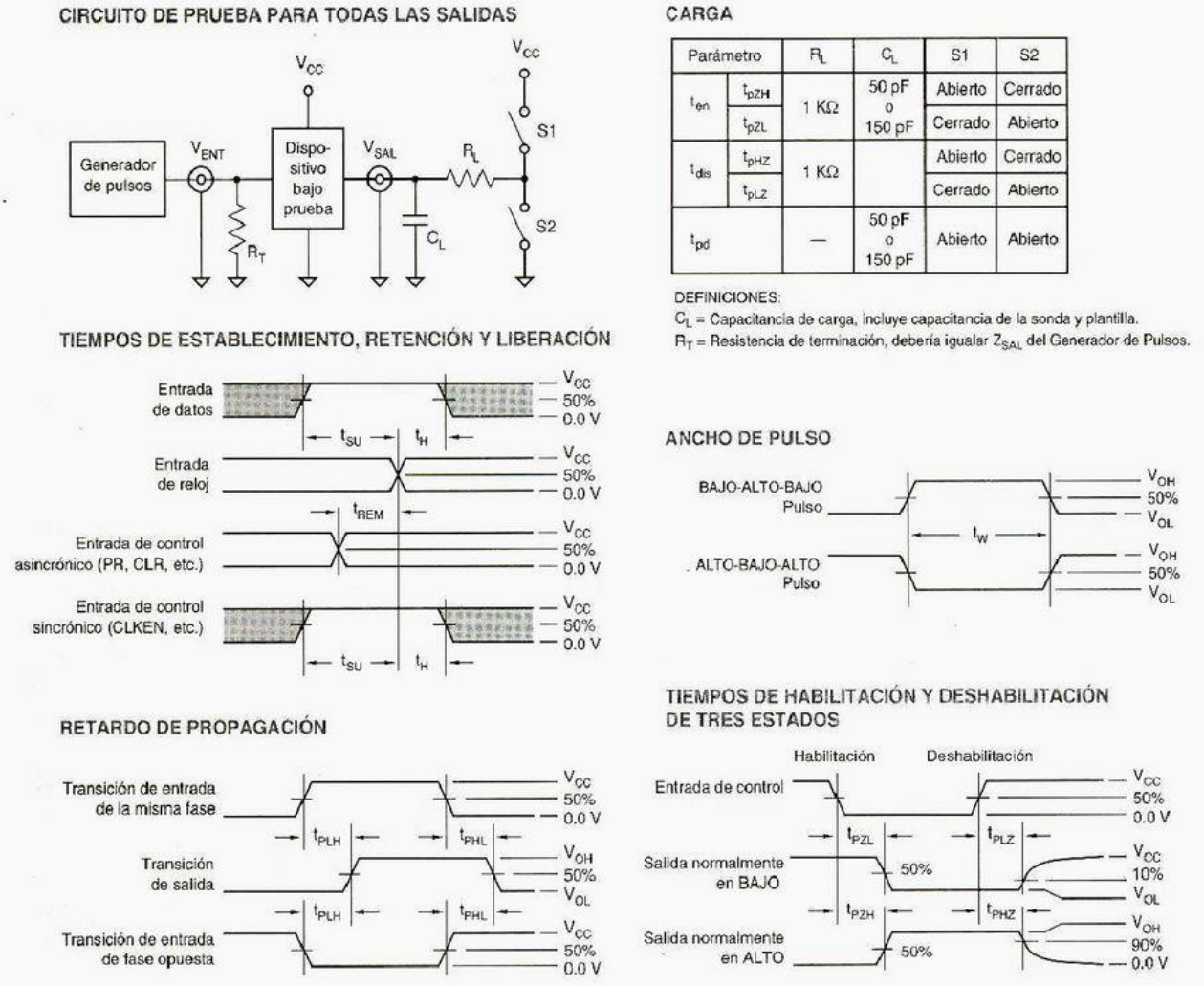


Figura 3-24 Circuitos de prueba y formas de onda para la lógica de la serie HC.

3.5 Comportamiento eléctrico de estado estable de los dispositivos CMOS

En esta sección analizaremos el comportamiento en estado estable de los circuitos CMOS, es decir, el comportamiento de los circuitos cuando no cambian sus entradas y salidas. La siguiente sección analiza el comportamiento dinámico, que incluye la velocidad y disipación de energía.

3.5.1 Niveles lógicos y márgenes de ruido

La tabla de la figura 3-10(b) de la página 88 define el comportamiento del inversor de CMOS cuando se aplican solamente dos voltajes discretos de entrada; otros voltajes

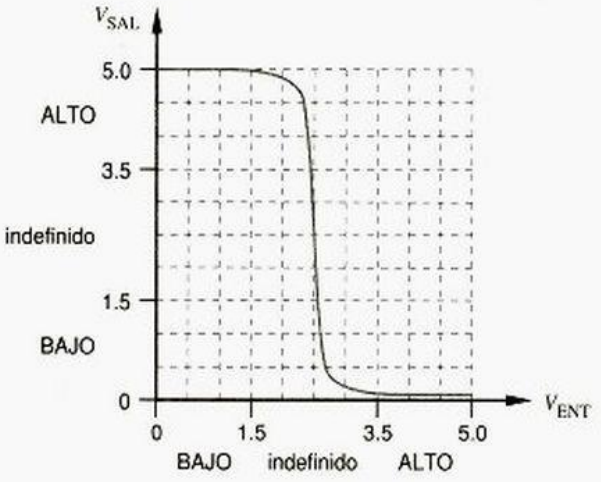


Figura 3-25
Característica de transferencia típica de entrada-salida para un inversor CMOS.

de entrada pueden producir diferentes voltajes de salida. La característica de transferencia entrada-salida completa puede describirse con la ayuda de una gráfica como la de la figura 3-25. En esta gráfica, el voltaje de entrada varía de 0 a 5 V, como se observa en el eje *X*; el eje *Y* representa el voltaje de salida.

Si confiamos en la curva de la figura 3-25, podríamos definir un nivel de entrada BAJO de CMOS como cualquier voltaje inferior a 2.4 V, y un nivel de entrada ALTO como cualquier voltaje superior a 2.6 V. Solamente cuando la entrada se encuentra entre 2.4 y 2.6 V el inversor produce un voltaje de salida no lógico, bajo esta definición.

Por desgracia, la característica típica de transferencia que se indica en la figura 3-25 es sólo eso: típica, no se puede garantizar. Ésta varía grandemente bajo diferentes condiciones de voltaje de alimentación, temperatura y carga de salida. La característica de transferencia puede inclusive variar dependiendo de la fecha de fabricación del dispositivo. Por ejemplo, después de meses de tratar de comprender por qué las compuertas que se fabricaron en una fecha particular no operaban adecuadamente, cuenta la leyenda que un fabricante descubrió que las compuertas defectuosas eran víctimas de contaminación aérea debido a un perfume particularmente pernicioso ¡que utilizó una de las operarias de la línea de producción!

La experiencia práctica de la ingeniería de sonido indica que utilicemos especificaciones más conservadoras para BAJO y ALTO. La figura 3-26 muestra las especificaciones conservadoras para una familia típica de lógica CMOS (serie HC). El fabricante de los dispositivos CMOS especifica estos parámetros en hojas de datos similares a las de la tabla 3-3, y se definen como sigue:

V_{IHmin} El mínimo voltaje de salida en el estado ALTO.
V_{IHmin} El mínimo voltaje de entrada que se garantiza será reconocido como ALTO.
V_{ILmax} El máximo voltaje de entrada que se garantiza será reconocido como BAJO.
V_{OLmax} El máximo voltaje de salida en el estado BAJO.

Los voltajes de entrada están determinados primordialmente por los umbrales de conmutación de ambos transistores, mientras que los voltajes de salida están determinados por la resistencia "de encendido" de los transistores.

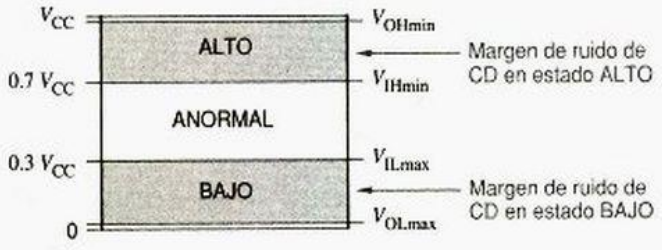


Figura 3-26
Niveles lógicos y márgenes de ruido para la familia lógica CMOS serie HC.

Todos los parámetros de la figura 3-26 están garantizados por los fabricantes de CMOS en un intervalo específico de temperatura y carga de salida. Los parámetros también están garantizados en un intervalo de voltajes de alimentación V_{CC}, típicamente de 5.0 V ± 10%.

La hoja de datos de la tabla 3-3 de la página 98 especifica valores de cada uno de estos parámetros para los CMOS de la serie HC. Nótese que existen dos valores especificados para V_{OHmin} y V_{OLmax}, dependiendo de si la corriente de salida (I_{OH} o I_{OL}) es pequeña o grande. Cuando las salidas del dispositivo se conectan a otras entradas CMOS, la corriente de salida es baja (por ejemplo, $I_{OL} \leq 20\ \mu A$), de modo que hay muy poca caída de voltaje a través de los transistores de salida. En las siguientes subsecciones nos enfocaremos a estas aplicaciones "puras" de los CMOS.

El voltaje de alimentación V_{CC} y el voltaje de tierra se denominan *líneas de alimentación*. Normalmente los niveles CMOS son una función de las líneas de alimentación:

líneas de alimentación

V_{OHmin}	$V_{CC} - 0.1$ V
V_{IHmin}	70% de V_{CC}
V_{ILmax}	30% de V_{CC}
V_{OLmax}	tierra + 0.1 V

Nótese en la tabla 3-3 que V_{OHmin} está especificado a 4.4 V. Esto es solamente una caída de 0.1 de V_{CC}, ya que el peor de los casos se especifica con V_{CC} a su valor mínimo de **5.0 – 10% = 4.5 V.**

El *margen de ruido de CD* indica cuánto ruido se necesita para corromper un voltaje de salida (en el peor de los casos) en un valor que no pueda ser reconocido adecuadamente por una entrada. Para los CMOS de la serie HC en el estado BAJO, V_{ILmax} (1.35 V) excede V_{OLmax} (0.1 V) por 1.25 V de modo que el margen de ruido de CD del estado BAJO es de 1.25 V. De igual forma, el margen de ruido de CD en el estado ALTO es de 1.25 V. En general, las salidas CMOS tienen excelentes márgenes de ruido de CD cuando excitan a otras entradas CMOS.

margen de ruido de CD

Sin importar el voltaje que se aplica en la entrada de un inversor CMOS, la entrada consume muy poca corriente, solamente se considera la corriente de fuga de las dos compuertas de los transistores. El valor máximo de corriente que puede fluir también se encuentra especificado por el fabricante del dispositivo:

I_{IH} La corriente máxima que fluye en la entrada en el estado ALTO.

I_{IL} La corriente máxima que fluye en la entrada en el estado BAJO.

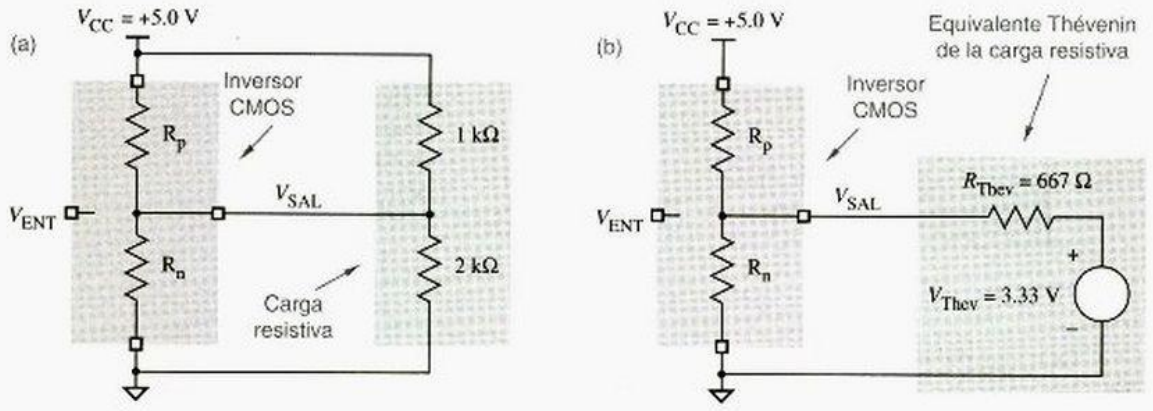


Figura 3-27 Modelo resistivo de un inversor CMOS con una carga resistiva: a) mostrando el circuito de carga real; b) haciendo uso del equivalente de Thévenin de la carga.

La corriente de entrada que se indica en la tabla 3-3 para el 'HC00 es tan sólo ±1 μA. De este modo, se requiere poca energía para mantener una entrada CMOS en un estado o en el otro. Éste es un constraste considerable con los circuitos bipolares de lógica como TTL y ECL, cuyas entradas pueden consumir una cantidad considerable de corriente (y energía) en uno o ambos estados.

3.5.2 Comportamiento del circuito con cargas resistivas

Como se mencionó con anterioridad, las entradas de las compuertas CMOS tienen una impedancia muy elevada y toman poca corriente de los circuitos que manejan. Sin embargo, existen otros dispositivos que requieren cantidades poco triviales de corriente para funcionar. Cuando un dispositivo de esta naturaleza se conecta a una salida CMOS, lo llamamos una *carga resistiva* o una *carga de CD*. Aquí tenemos algunos ejemplos de cargas resistivas:

carga resistiva
carga de CD

- Se pueden incluir resistores discretos para proporcionar una terminación en la línea de transmisión, como se analiza en la sección 11.4.
- En realidad los resistores discretos pueden no estar presentes en el circuito, pero la carga que presentan uno o más dispositivos TTL u otras entradas que no sean CMOS pueden modelarse como una red sencilla de resistores.
- Los resistores pueden ser parte de o pueden modelar un dispositivo que consume energía, como en el caso de un diodo emisor de luz (LED, por sus siglas en inglés) o la bobina de un relevador.

Cuando se conecta la salida de un circuito CMOS a una carga resistiva, el comportamiento de la salida no se acerca tanto a la situación ideal que hemos descrito con anterioridad. En cualquier estado lógico, el transistor de salida CMOS que se encuentra "encendido" tiene una resistencia distinta de cero, y cualquier carga que se conecte en la terminal de salida ocasionará una caída de voltaje a través de esta resistencia. De este modo, en el estado BAJO, el voltaje de salida puede ser algo mayor de 0.1 V, y en el estado ALTO puede ser menor de 4.4 V. La manera más fácil de observar lo anterior consiste en examinar un modelo resistivo del circuito y la carga CMOS.

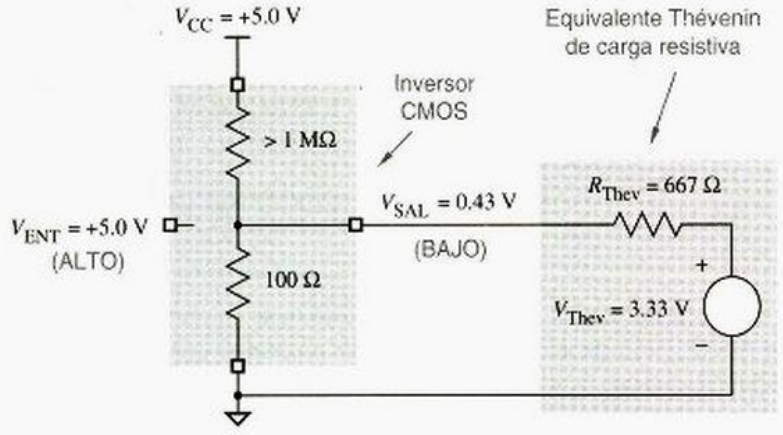


Figura 3-28
Modelo resistivo de salida BAJO de CMOS con carga resistiva.

La figura 3-27(a) muestra el modelo resistivo. Los transistores de canal-*p* y canal-*n* tienen las resistencias R_p y R_n, respectivamente. En operación normal, una resistencia es alta (>1 MΩ) y la otra es baja (quizás 100 Ω), dependiendo de si el voltaje de entrada es ALTO o BAJO. La carga en este circuito se compone de dos resistores que se conectan a las líneas de alimentación; un circuito real puede tener cualquier cantidad de resistores, o incluso una red resistiva más compleja. En cualquier caso, una carga resistiva que esté formada en su totalidad por resistores y fuentes de voltaje, siempre podrá modelarse por una red equivalente de Thévenin, como la que se ilustra en la figura 3-27(b).

Cuando el inversor CMOS tiene una entrada ALTO, la salida debería ser BAJO; el voltaje real de salida puede predecirse utilizando el modelo resistivo que se indica en la figura 3-28. El transistor de canal-*p* se encuentra "apagado" y tiene una resistencia muy elevada, suficientemente alta para que sea despreciable en los siguientes cálculos. El tran-

RECORDEMOS A THÉVENIN

Cualquier circuito de dos terminales consistente de solamente fuentes de voltaje y resistores puede ser modelado por un *equivalente de Thévenin* que consta de una sola fuente de voltaje en serie con un resistor simple. El *voltaje de Thévenin* es el voltaje en circuito abierto del circuito original, y la *resistencia de Thévenin* es el voltaje de Thévenin dividido entre la corriente de cortocircuito del circuito original.

En el ejemplo de la figura 3-27, el voltaje Thévenin de la carga resistiva, incluyendo su conexión a V_{CC}, se establece por los resistores de 1 kΩ y 2 kΩ, los cuales forman un divisor de voltaje:

$$V_{Thev} = \frac{2\ \text{k}\Omega}{2\ \text{k}\Omega + 1\ \text{k}\Omega} \cdot 5.0\ \text{V} = 3.33\ \text{V}$$

La corriente de cortocircuito es (5.0 V)/(1 kΩ) = 5 mA, de modo que la resistencia Thévenin es (3.3 V)/(5 mA) = 667 Ω. Los lectores experimentados pueden reconocer esto como la resistencia en paralelo de los resistores de 1 kΩ y 2 kΩ.

sistor de canal-*n* se encuentra "encendido" y tiene una resistencia baja, que suponemos es 100 Ω (La resistencia real "de encendido" depende de la familia CMOS y otras características tales como temperatura de operación, etc.). El transistor "encendido" y el resistor equivalente de Thevenin R_{Thev} de la figura 3-28 forman un divisor de voltaje sencillo. El voltaje de salida se puede calcular como se indica:

$$\begin{aligned} V_{SAL} &= 3.33\ V \cdot [100/(100 + 667)] \\ &= 0.43\ V \end{aligned}$$

En forma semejante, cuando el inversor tiene una entrada BAJO, la salida debería ser ALTO, y el voltaje real de salida puede predecirse con el modelo de la figura 3-29. Supondremos que la resistencia "de encendido" del transistor de canal-*p* es 200 Ω. Una vez más, el transistor "encendido" y el resistor equivalente de Thévenin R_{Thev} en la figura forman un divisor de voltaje, y el voltaje de salida resultante puede calcularse como se explica a continuación:

$$\begin{aligned} V_{SAL} &= 3.33\ V + (5\ V - 3.33\ V) \cdot [667/(200 + 667)] \\ &= 4.61\ V \end{aligned}$$

En la práctica, rara vez se necesita calcular voltajes de salida como en los ejemplos anteriores. De hecho, es normal que los fabricantes de CIs no especifiquen las resistencias equivalentes de los transistores "encendidos", de modo que no se tendría la información necesaria para realizar los cálculos todas formas. En vez de eso, los fabricantes de CI especifican la carga máxima para la salida en cada estado (ALTO o BAJO), y garantizan un voltaje de salida —en el peor de los casos— para esa carga. La carga se especifica en términos de corriente:

I_{OLmax} La corriente máxima que puede consumir la salida en el estado BAJO mientras mantiene un voltaje de salida inferior a V_{OLmax}.

I_{OHmax} La corriente máxima que puede suministrar la salida en el estado ALTO mientras mantiene un voltaje de salida superior a V_{OHmin}.

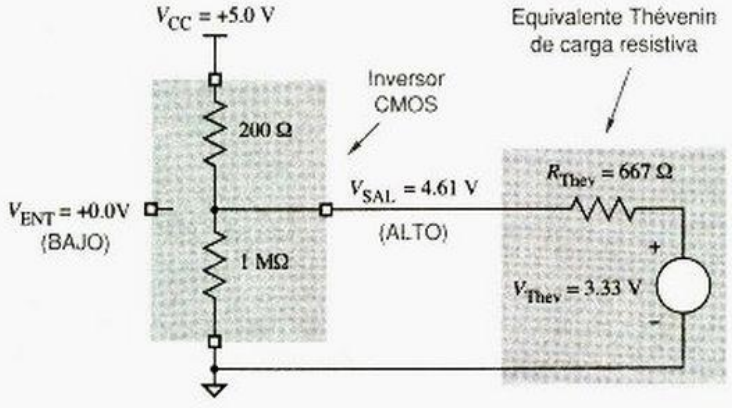


Figura 3-29
Modelo resistivo para salida ALTO CMOS con carga resistiva.

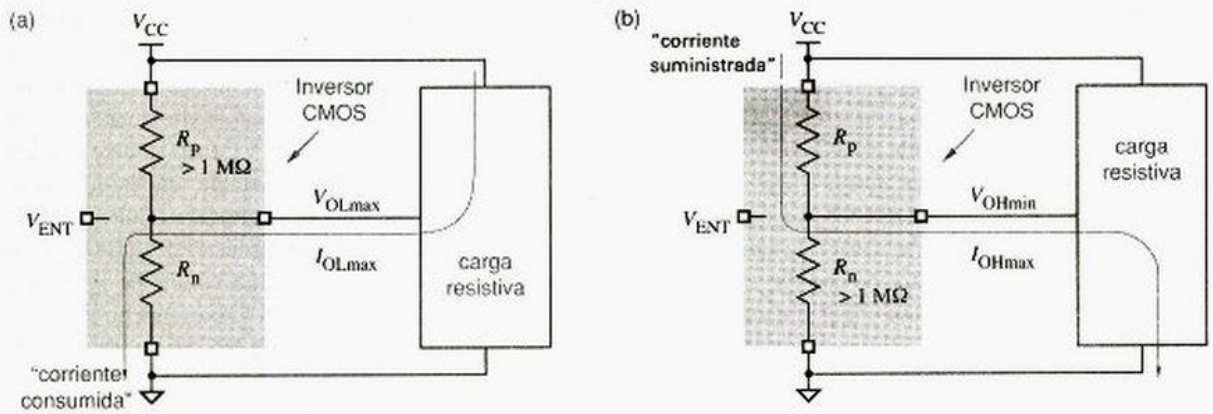


Figura 3-30 Definiciones de circuito de (a) I_{OLmax}; (b) I_{OHmax}.

Estas definiciones se ilustran en la figura 3-30. Se dice que la salida de un dispositivo *consume corriente* cuando la corriente fluye desde la fuente de alimentación, a través de la carga y a través de la salida del dispositivo hacia tierra como en (a). Se dice que la salida *suministra corriente* cuando la corriente fluye desde la fuente de alimentación, fuera de la salida del dispositivo y a través de la carga hacia tierra como en (b).

corriente consumida

corriente suministrada

La mayor parte de los dispositivos CMOS tiene dos conjuntos de especificaciones de carga. Un conjunto es para "cargas CMOS", donde la salida del dispositivo se conecta a otras entradas CMOS, que consumen poca corriente. El otro conjunto es para "cargas TTL", donde la salida se conecta a cargas resistivas tales como entradas TTL u otros dispositivos que consumen una cantidad considerable de corriente. Por ejemplo, las especificaciones para las salidas CMOS de la serie HC se mostraron en la tabla 3-3 y se repiten en la tabla 3-4.

Conviene indicar que en la tabla la corriente de salida en el estado ALTO se indica como una cantidad negativa. Por convención, el *flujo de corriente* que se mide en la terminal de un dispositivo es positivo si la corriente positiva fluye *hacia el interior* del dispositivo; en el estado ALTO, la corriente fluye *hacia fuera* de la terminal de salida.

flujo de corriente

Tabla 3-4 Especificaciones de carga de salida para CMOS serie HC con una alimentación de 5 V.

	Carga CMOS		Carga TTL	
Parámetro	**Nombre**	**Valor**	**Nombre**	**Valor**
Corriente máxima de salida en estado BAJO (mA)	I_{OLmaxC}	0.02	I_{OLmaxT}	4.0
Voltaje máximo de salida en estado BAJO (V)	V_{OLmaxC}	0.1	V_{OLmaxT}	0.33
Corriente máxima de salida en estado ALTO (mA)	I_{OHmaxC}	−0.02	I_{OHmaxT}	−4.0
Voltaje mínimo de salida en estado ALTO (V)	V_{OHminC}	4.4	V_{OHminT}	3.84

Como se muestra en la tabla, con cargas CMOS, el voltaje de salida de la compuerta CMOS se mantiene dentro de 0.1 V de la línea de alimentación. Con cargas TTL, el voltaje de salida puede degradarse bastante. Cabe indicar que para la misma corriente de salida (±4 mA), la caída máxima de voltaje con respecto a la línea de alimentación es el doble en el estado ALTO (0.66 V) que en el estado BAJO (0.33 V). Esto sugiere que los transistores CMOS de canal-*p* de la serie HC tienen una resistencia de "encendido" más alta que los transistores de canal-*n*. Esto es natural, porque en cualquier circuito CMOS, un transistor de canal-*p* tiene más de dos veces la resistencia de "encendido" que un transistor de canal-*n* con la misma área. Podrían obtenerse caídas de voltaje de la misma magnitud en ambos estados si las dimensiones de los transistores fueran mucho mayores en los transistores de canal-*p* que en los transistores de canal-*n*, pero por varias razones éste no es el caso.

Se puede utilizar la ley de Ohm para determinar cuánta corriente suministra o consume una salida en una situación determinada. En la figura 3-28 de la página 103, el transistor de canal-*n* "encendido", que está modelado por el resistor de 100 Ω, tiene una caída de 0.43 V a través de él; por consiguiente, consume (0.43 V)/(100 Ω) = 4.3 mA de corriente. En forma similar, el transistor de canal-*p* "encendido" de la figura 3-29 suministra (0.39 V)/(200 Ω) = 1.95 mA.

En general, no se publican las resistencias reales de "encendido" de los transistores de salida CMOS, de modo que no siempre es posible emplear los modelos exactos de los párrafos anteriores. Sin embargo, usted puede estimar las resistencias de "encendido" utilizando las ecuaciones siguientes, las cuales se basan en especificaciones que siempre se publican:

$$R_{\text{p(encendido)}} = \frac{V_{\text{DD}} - V_{\text{OHminT}}}{|I_{\text{OHmaxT}}|}$$

$$R_{\text{n(encendido)}} = \frac{V_{\text{OLmaxT}}}{I_{\text{OLmaxT}}}$$

Estas ecuaciones emplean la ley de Ohm para calcular la resistencia "de encendido" como la caída de voltaje a través del transistor "encendido" dividida entre la corriente que circula por el mismo, con una carga resistiva en el peor de los casos. Haciendo uso de los datos del CMOS serie HC de la tabla 3-4, podemos calcular $R_{\text{p(encendido)}} = 175\ \Omega$ y $R_{\text{n(encendido)}} = 82.5\ \Omega$.

Pueden hacerse buenas estimaciones de la corriente de salida *en el peor de los casos*, suponiendo que *no existe* caída de voltaje a través del transistor "encendido". Esta suposición simplifica el análisis, y produce un resultado conservador que casi siempre es bastante bueno para fines prácticos. Por ejemplo, la figura 3-31 muestra un inversor CMOS que excita a la misma carga equivalente de Thévenin que utilizamos en los ejemplos anteriores. No se muestra el modelo resistivo de la estructura de salida porque ya no se necesita. Suponemos que no existe caída de voltaje a través del transistor CMOS que está "encendido". En (a), con la salida BAJO, todo el voltaje del equivalente de Thévenin (3.33 V) aparece a través de R_{Thev} y el valor estimado para la corriente consumida es (3.33 V)/(667 Ω) = 5.0 mA. En (b), con la salida ALTO y suponiendo una alimentación de 5.0 V, la caída de voltaje a través de R_{Thev} es 1.67 V y el valor estimado para la corriente de suministro es (1.67 V)/(667 Ω) = 2.5 mA.

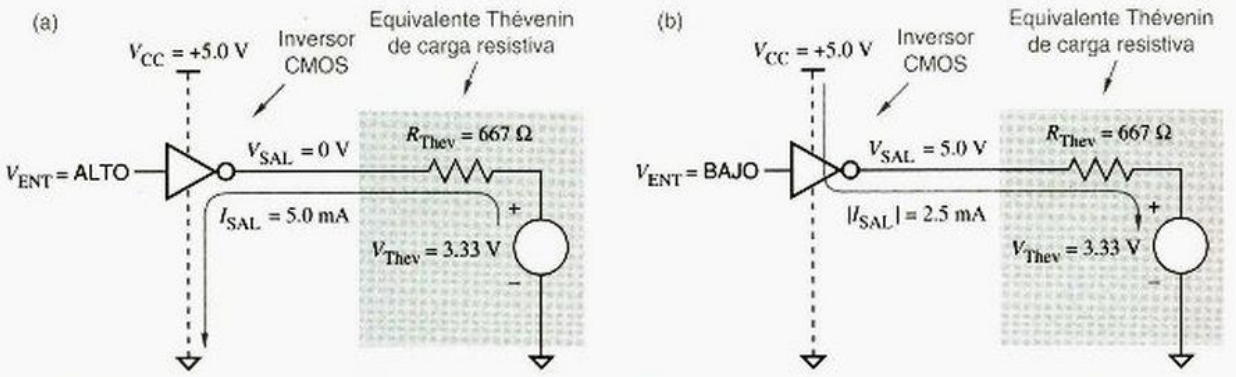


Figura 3-31 Estimación de corriente consumida y suministrada: a) salida en BAJO; b) salida en ALTO.

Una característica importante del inversor CMOS (o de cualquier circuito CMOS) es que la estructura de salida por sí misma consume poca corriente en cualquier estado, ALTO o BAJO. En cualquier estado, uno de los transistores se encuentra en el estado "apagado" de alta impedancia. Todo el flujo de corriente del que hemos estado hablando se presenta cuando una carga resistiva se conecta a la salida CMOS. Si no hay carga, entonces no existe flujo de corriente, y el consumo de energía es cero. Sin embargo, cuando se conecta una carga, la corriente fluye tanto por la carga como por el transistor "encendido" y ambos consumen energía.

LA VERDAD ACERCA DEL CONSUMO DE ENERGÍA

Como lo hemos establecido en otra parte, la resistencia de un transistor "apagado" se encuentra arriba de un megohm, pero no es infinita. Por consiguiente, circula en realidad una muy diminuta corriente de fuga en los transistores "apagados", y la estructura de salida CMOS tiene un correspondiente consumo diminuto de energía distinto de cero. En la mayoría de las aplicaciones, este consumo de energía es lo suficientemente diminuto para ignorarse. Por lo general sólo es significativo en el "modo de espera" en dispositivos alimentados por baterías, como la computadora portátil en la cual este capítulo se preparó por primera vez.

3.5.3 Comportamiento del circuito con entradas no ideales

Hasta ahora, hemos asumido que las entradas ALTO y BAJO de un circuito CMOS son voltajes ideales, muy cercanos a las líneas de alimentación. Sin embargo, el comportamiento de un circuito inversor real CMOS depende tanto del voltaje de entrada como de las características de la carga. Si el voltaje de entrada no se encuentra cerca de la línea de alimentación, entonces el transistor "encendido" puede no encontrarse completamente "encendido" y su resistencia puede aumentar. Del mismo modo, el transistor "apagado" puede no estar completamente "apagado" y su resistencia puede ser inferior a un megohm. Estos dos efectos se combinan para desplazar el voltaje de salida lejos de la línea de alimentación.

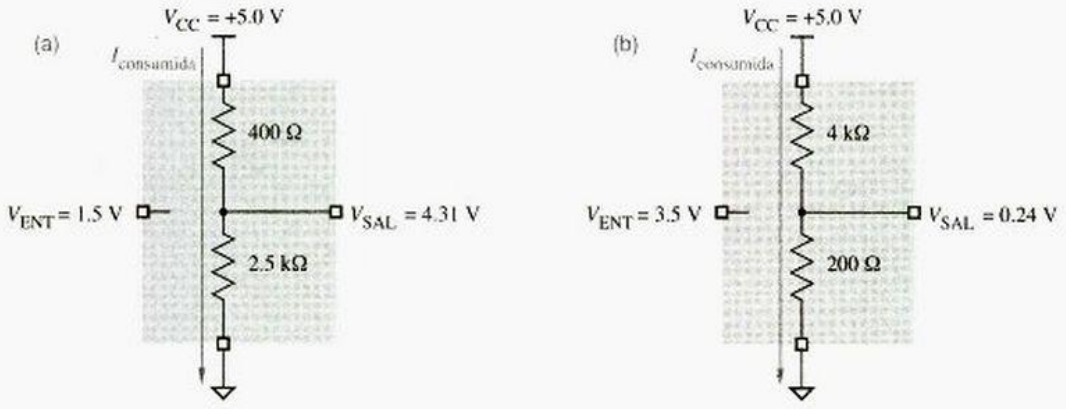


Figura 3-32 Inversor CMOS con voltajes de entrada no ideales: a) circuito equivalente con entrada de 1.5 V; b) circuito equivalente con entrada de 3.5 V.

Por ejemplo, la figura 3-32(a) muestra un comportamiento posible del inversor CMOS que tiene una entrada de 1.5 V. La resistencia del transistor de canal-*p* se ha duplicado en este punto, y la resistencia del transistor de canal-*n* comienza a activarse. (Estos valores son meras suposiciones para propósitos de ejemplo; los valores reales dependen de las características específicas de los transistores.)

En la figura, la salida de 4.31 V todavía se encuentra en el intervalo de validez para una señal ALTO, pero no se acerca al valor ideal de 5.0 V. En forma semejante, con una entrada de 3.5 V en (b), la salida BAJO es de 0.24 V, no de 0 V. En general, esta pequeña degradación en el voltaje de salida es tolerable; lo preocupante es que la estructura de salida comsume una cantidad nada despreciable de energía. El flujo de corriente con la entrada de 1.5 V es de

$$I_{\text{consumida}} = 5.0 \text{ V}/(400\ \Omega + 2.5\ \text{k}\Omega) = 1.72 \text{ mA}$$

y el consumo de energía es

$$P_{\text{consumida}} = 5.0 \text{ V} \cdot I_{\text{consumida}} = 8.62 \text{ mW}$$

El voltaje de salida de un inversor CMOS se deteriora aún más con una carga resistiva. Una carga de este tipo puede existir por cualquiera de las razones que se analizaron con anterioridad. La figura 3-33 muestra un comportamiento posible del inversor CMOS que tiene una carga resistiva. Con una entrada de 1.5 V, la salida de 3.98 V todavía se encuentra dentro del intervalo de validez para una señal ALTO, pero está muy lejos del valor ideal de 5.0 V. De manera similar, con una entrada de 3.5 V como se muestra en la figura 3-34, la salida BAJO es de 0.93 V, y no de 0 V.

En sistemas CMOS "puros", todos los dispositivos lógicos del circuito son CMOS. Puesto que las entradas CMOS tiene una impedancia muy elevada, presentan poca carga resistiva a las salidas CMOS que las activan. Por lo tanto, los niveles de salida CMOS permanecen muy cerca de las líneas de alimentación (0 V y 5 V). Y ninguno de los dispositivos desperdicia energía en sus estructuras de salida. Por otra parte, si las salidas TTL u otras señales lógicas no ideales se conectan a las entradas CMOS, entonces las salidas

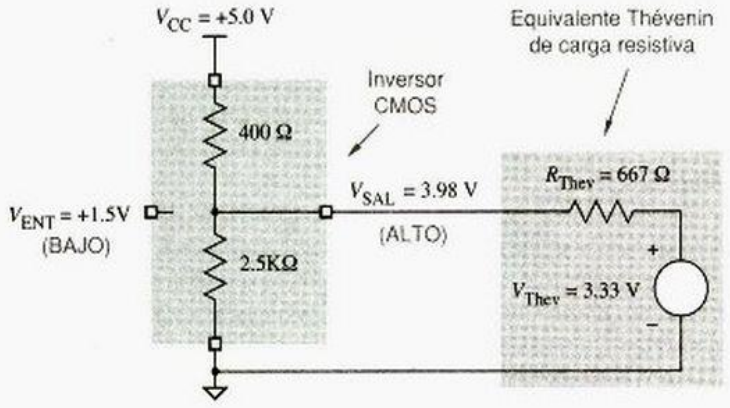


Figura 3-33
Inversor CMOS con carga y entrada no ideal de 1.5 V.

CMOS utilizan energía como se indica en esta subsección; esto se formaliza en el cuadro que está en la parte superior de la página 139. Además, si las entradas TTL u otras cargas resistivas se conectan a las salidas CMOS, entonces las salidas CMOS utilizan la energía como se indica en la subsección anterior.

3.5.4 Fan out

fan out

El *Fan out* de una compuerta lógica es la cantidad de entradas que puede controlar la compuerta sin exceder sus especificaciones de carga en el peor de los casos. El *fan out* depende no solamente de las características de la salida, sino también de las entradas que se estén controlando. El *Fan out* debe analizarse considerando los estados posibles de salida, ALTO y BAJO.

Por ejemplo, como se indica en la tabla 3-4 de la página 105, la corriente máxima de salida de estado BAJO I_{OLmaxC} para una compuerta CMOS serie HC que excita a entradas CMOS es de 0.02 mA (20 μA). También establecimos con anterioridad que la máxima corriente de entrada I_{Imax} para una entrada CMOS serie HC en cualquier estado es ±1 μA. Por lo tanto, el *fan out de estado BAJO* de una salida de CMOS serie HC que controla entradas CMOS de la serie HC es 20. La tabla 3-4 muestra también que la corriente máxima de salida del estado ALTO I_{OHmaxC} es −0.02 mA (−20 μA).

fan out de estado BAJO

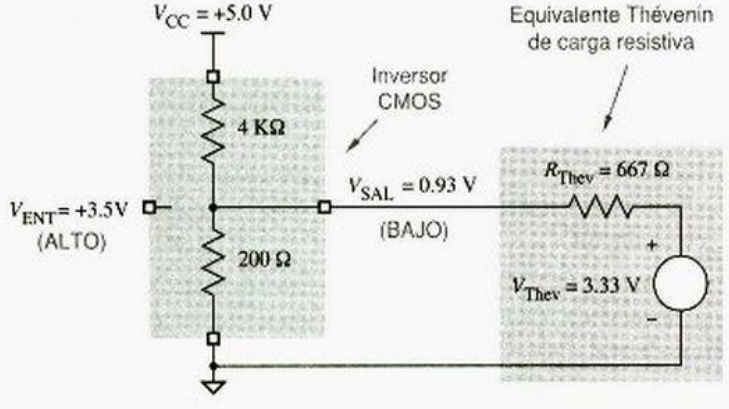


Figura 3-34
Inversor CMOS con carga y entrada no ideal de 3.5 V.

fan out de estado ALTO

Por consiguiente, *el fan out de estado ALTO* para una salida CMOS de la serie HC que excita entradas de la serie HC también es 20.

fan out total

Nótese que el fan out estado ALTO y de estado BAJO de una compuerta no son necesariamente iguales. En general, *el fan out total* de una compuerta es el mínimo de el fan out de estado ALTO y de el fan out de estado BAJO, es decir, 20 para el ejemplo anterior.

En el ejemplo de fan out que acabamos de terminar, asumimos que necesitábamos mantener la salida de la compuerta en los niveles CMOS, es decir, dentro de 0.1 V del voltaje de las líneas de alimentación. Si estuviéramos dispuestos a tolerar algo de degradación, en niveles de salida TTL, entonces podríamos utilizar I_{OLmaxT} e I_{OHmaxT} para el cálculo del fan out. La tabla 3-4 muestra que estas especificaciones son 4.0 mA y –4.0 mA, respectivamente. Por lo tanto, el fan out de una salida CMOS de la serie HC que excita entradas de la serie HC a niveles TTL es 4000, al parecer, virtualmente ilimitada.

fan out CD

Bueno, en realidad no es así. Los cálculos que acabamos de realizar proporcionan *el fan out de CD*, definida como el número de entradas que puede controlar una salida, cuando el *estado de la salida permanece constante* (ALTO o BAJO). Incluso si se satisface la especificación de fan out de CD, una salida CMOS que controla un gran número de entradas puede no comportarse satisfactoriamente en las transiciones, BAJO a ALTO, o viceversa.

Durante las transiciones, la salida CMOS debe cargar o descargar la capacitancia parásita que está asociada con las entradas que maneja. Si esta capacitancia es demasiado grande, la transición de BAJO a ALTO (o viceversa) puede ser demasiado lenta, provocando un funcionamiento inapropiado del sistema.

fan out de CA

La habilidad de una salida para cargar y descargar la capacitancia parásita se conoce también como *fan out de CA*, aunque rara vez se calcula en forma tan precisa como el fan out de CD. Como se verá en la sección 3.6.1, es más un asunto de decidir cuánta degradación en la velocidad se va a tolerar.

3.5.5 Efectos de la carga

Al sobrecargar una salida más allá de su fan out se producen los siguientes efectos:

- En el estado BAJO, el voltaje de salida (V_{OL}) puede incrementarse más allá de V_{OLmax}.
- En el estado ALTO, el voltaje de salida (V_{OH}) puede caer por debajo de V_{OHmin}.
- El retardo de propagación de la salida puede aumentar más allá de las especificaciones.
- Los tiempos de ascenso y descenso de salida pueden incrementarse más allá de sus especificaciones.
- La temperatura de operación del dispositivo puede incrementarse, reduciendo por ello la confiabilidad del dispositivo y con el paso del tiempo puede ocasionar fallas en el mismo.

Los primeros cuatro efectos reducen los márgenes de ruido de CD y los márgenes de temporización del circuito. De este modo, un circuito ligeramente sobrecargado puede funcionar en forma adecuada bajo condiciones ideales, pero la experiencia nos dice que llegará a fallar una vez que se encuentre fuera del ambiente agradable del laboratorio de ingeniería.

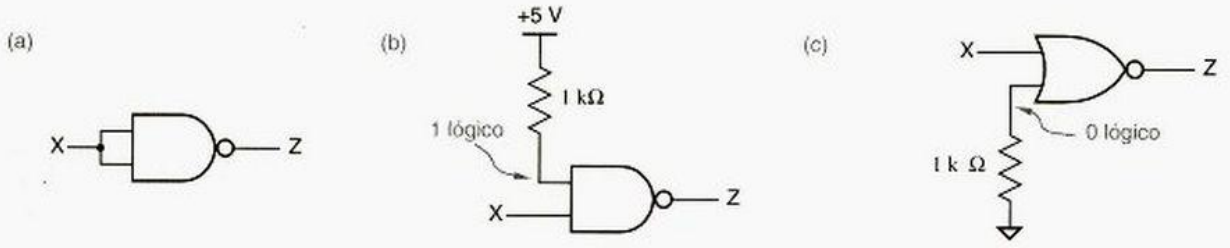


Figura 3-35 Entradas sin utilizar: a) unidas a otra entrada; b) NAND levantada; c) NOR disminuida.

3.5.6 Entradas que no se utilizan

En ocasiones, no se utilizan todas las entradas de una compuerta lógica. En un problema real de diseño se puede necesitar una compuerta con n entradas pero solamente hay disponible una compuerta con $(n + 1)$ entradas. Al unir dos entradas de la compuerta con $(n + 1)$ entradas se obtiene la funcionalidad de una compuerta de n entradas. Convénzase usted mismo de este hecho de manera intuitiva ahora, o haga uso del álgebra de conmutación para probarlo después de estudiar la sección 4.1. La figura 3-35(a) muestra una compuerta NAND con sus entradas unidas.

También se pueden unir las entradas que no se utilizan a un valor lógico constante. Una entrada AND o NAND sin utilizar debería conectarse al 1 lógico, como en (b), y una entrada OR o NOR sin utilizar debería conectarse al 0 lógico, como en (c). En el diseño de circuitos de alta velocidad es preferible emplear el método (b) o (c) en lugar del método (a), el cual incrementa la carga capacitiva sobre la señal de control y puede alentar las cosas. En (b) y (c), se utiliza un resistor con valor de 1 a 10 kΩ, y un solo resistor (pull-up o pull-down) puede manejar a las entradas que no se utilizan. También se pueden unir directamente las entradas que no se utilizan a la línea de alimentación adecuada.

entrada flotante

Las entradas CMOS que no se utilizan nunca deben estar flotando o sin conectarse a ningún punto. Cualquier entrada que esté flotando se comportará como si tuviera una señal BAJO y normalmente mostrará un valor de 0 V cuando se examine con un osciloscopio o voltímetro. De modo que se puede pensar que una entrada OR o NOR que no se utiliza puede dejarse flotando, porque actuará como si se le aplicara un 0 lógico y no afectará la salida de la compuerta. Sin embargo, puesto que las entradas CMOS tienen una alta impedancia, solamente se necesita una pequeña cantidad de ruido en el circuito

ERRORES SUTILES La entradas CMOS flotantes son con frecuencia las causantes de misteriosos comportamientos de los circuitos, a medida que una entrada sin utilizar cambia erráticamente su estado efectivo basado en ruido y condiciones en otra parte del circuito. Cuando se intenta depurar un problema de este tipo, la capacitancia extra de una punta de prueba de osciloscopio tocando la entrada flotante a menudo es suficiente para amortiguar el ruido y hacer que el problema desaparezca. Esto puede ser especialmente desconcertante y difícil de resolver, ¡si usted no se da cuenta de que la entrada se encuentra flotando!

para que una entrada flotante parezca tener el nivel ALTO (en forma temporal), esto puede generar ciertas fallas intermitentes y desagradables en el circuito.

3.5.7 Picos de corriente y capacitores de desacoplamiento

picos de corriente

Cuando una salida CMOS conmuta entre los estados BAJO y ALTO, la corriente fluye desde V_{CC} a tierra a través de los transistores de canal-*p* y canal-*n* que están encendidos en forma parcial. Estas corrientes —que también se conocen como *picos de corriente* a consecuencia de su breve duración—, pueden aparecer como ruido en las conexiones de la fuente de alimentación o en las conexiones de tierra de un circuito CMOS, en especial cuando conmutan simultáneamente varias salidas.

capacitores de desacoplamiento

capacitores de filtrado

Por esta razón los sistemas que utilizan circuitos CMOS requieren el uso de *capacitores de desacoplamiento* entre V_{CC} y tierra. Estos capacitores deben distribuirse a lo largo del circuito, por lo menos uno cada pulgada o por chip, para suministrar corriente durante las transiciones de nivel. Los *capacitores de filtrado* de alto valor que se encuentran (por lo general) en la misma fuente de alimentación no satisfacen este requerimiento, porque la inductancia parásita del alambrado evita que suministren la corriente con la rapidez necesaria, de ahí la necesidad de un sistema *físicamente distribuido* de capacitores de desacoplamiento.

3.5.8 Cómo destruir un dispositivo CMOS

Golpéelo con un martillo o simplemente camine a través de una alfombra y a continuación toque una terminal de entrada con su dedo. Debido a que las entradas del dispositivo CMOS tienen una impedancia muy alta, están expuestas a daños debido a las *descargas electrostáticas (ESD, electrostatic discharge)*.

descarga electrostática (ESD)

Las descargas electrostáticas se generan cuando la carga que se ha acumulado sobre una superficie pasa a través de un material dieléctrico hacia otra superficie que tiene carga opuesta. En el caso de la terminal de entrada de un dispositivo CMOS, el dieléctrico es el aislante que existe entre la terminal de entrada (compuerta) del transistor y las terminales de fuente o drenaje. Las descargas electrostáticas pueden dañar este aislamiento y ocasionar con ello un cortocircuito entre la entrada y la salida del dispositivo.

Las estructuras de entrada de los dispositivos CMOS modernos utilizan varias técnicas para reducir su susceptibilidad al daño que generan las descargas electrostáticas, pero ningún dispositivo es completamente inmune a ellas. Por lo tanto, para proteger los dispositivos CMOS contra las descargas electrostáticas durante las operaciones de ensamble y empaque, los fabricantes empacan los dispositivos en bolsas, carrilleras, tubos o espuma plástica conductora. Para evitar daños durante el manejo de los dispositivos CMOS, el personal técnico y de ensamble utiliza cintas o muñequeras conductoras que se conectan a tierra física por medio de un cable flexible; esto previene la acumulación de cargas estáticas en sus cuerpos a medida que se desplazan en las líneas de ensamble o laboratorio.

latch-up

Una vez que se instala el dispositivo CMOS en un sistema o circuito, otra fuente probable de daños es el *latch-up*. La estructura física de entrada de casi cualquier dispositivo CMOS incluye transistores bipolares parásitos entre V_{CC} y tierra que están configurados como un "rectificador controlado de silicio (SCR, siliconcontrolled rectifier)". En operación normal, este "SCR parásito" no tiene ningún efecto sobre el funcionamiento del dispositivo. Sin embargo, cuando el voltaje de entrada es menor a tierra o mayor que V_{CC} se puede "disparar" el SCR, creando un cortocircuito virtual entre V_{CC} y tierra. Una vez que se ha disparado el SCR, la única manera de apagarlo es apagar la

¡ELIMINE EL COMPORTAMIENTO RUDO, TRAUMATIZANTE!

Algunos ingenieros en diseño se consideran a sí mismos estar por encima de estos inconvenientes, pero para estar seguros usted debería tener las siguientes precauciones acerca de la ESD en el laboratorio:

- Antes de manipular un dispositivo CMOS, toque el gabinete metálico o chasis aterrizado de un instrumento conectado u otra fuente de tierra física.
- Antes de transportar un dispositivo CMOS, insértelo en espuma conductiva.
- Cuando transporte un tablero de circuito que contenga dispositivos CMOS, maneje el tablero por los bordes, y toque una terminal de tierra en el tablero a tierra física antes de andar paseando con él.
- Cuando le pase un dispositivo CMOS a un compañero, especialmente en un día seco de invierno, toque primero al compañero. Él o ella se lo agradecerán.

fuente de alimentación. Antes de que usted tenga una oportunidad de hacer esto, el SCR puede disipar una cantidad excesiva de energía que lo destruirá (en este caso usted puede ver el humo que sale del dispositivo.)

Un posible disparador del latch-up es el "subimpulso" negativo en las transiciones de señal de alta velocidad de ALTO a BAJO, que se analizan en la sección 11.4. En esta situación, la señal de entrada puede estar varios volts debajo del nivel de tierra durante ciertos nanosegundos antes de estabilizarse en el intervalo BAJO normal. Sin embargo, los circuitos modernos de lógica CMOS se fabrican con estructuras especiales que previenen los latch-up en este caso transitorios.

El latch-up puede ocurrir cuando las entradas CMOS son excitadas por las salidas de otro sistema o subsistema que tiene una fuente de alimentación independiente. Si una entrada ALTO se aplica a una compuerta CMOS antes de que esté presente la energía de alimentación, la compuerta puede llegar al estado de "latch-up" cuando se aplica la energía. De nueva cuenta, los circuitos lógicos CMOS modernos se fabrican con estructuras especiales que evitan esto en la mayoría de los casos. Sin embargo, si la salida que excita las entradas CMOS es capaz de suministrar cantidades considerables de corriente (por ejemplo, decenas de mA), es muy probable que suceda el latch-up. Una solución a este problema es aplicar energía antes de conectar los cables de entrada.

3.6 Comportamiento eléctrico dinámico de los dispositivos CMOS

Tanto la velocidad como el consumo de energía de un dispositivo CMOS dependen en gran parte de las características dinámicas (o de CA) del dispositivo y su carga, es decir, lo que sucede cuando la salida cambia entre los estados lógicos. Como parte del diseño interno de los ASIC CMOS, los diseñadores lógicos deben examinar con sumo cuidado los efectos de la carga de salida y modificar el diseño cuando la carga sea muy grande. Incluso en el diseño a nivel tarjeta, se deben considerar los efectos de la carga en las señales de reloj, bus, y otras señales que tienen un fan-out elevada o interconexiones largas.

La velocidad depende de dos características, el tiempo de transición y el retardo de propagación, que se analizarán en las subsecciones siguientes. La disipación de energía se estudia en la subsección 3.6.3

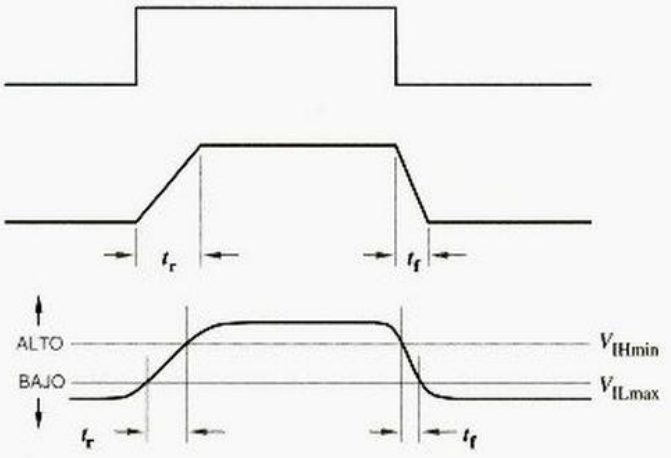


Figura 3-36
Tiempos de transición: a) caso ideal de conmutación en tiempo cero; b) una aproximación más realista; c) temporización real, mostrando los tiempos de ascenso y caída.

3.6.1 Tiempo de transición

tiempo de transición

La cantidad de tiempo que requiere la salida de un circuito lógico para cambiar de un estado a otro se conoce como *tiempo de transición*. La figura 3-36(a) muestra la situación ideal para el cambio de estado en las salidas: en tiempo cero. Sin embargo, las salidas reales no pueden cambiar instantáneamente, porque necesitan tiempo para cargar la capacitancia parásita del alambrado y otros componentes que controlan. Una visión más realista de una salida del circuito se muestra en (b). Una salida necesita cierta cantidad de tiempo, denominado *tiempo de ascenso* (t_r) para cambiar de BAJO a ALTO y un tiempo posiblemente diferente, denominado el *tiempo de caída* (t_f) para cambiar de ALTO a BAJO.

tiempo de ascenso (t_r)
tiempo de caída (t_f)

Incluso la figura 3-36(b) no es bastante precisa, porque la razón de cambio del voltaje de salida no cambia instantáneamente. En su lugar, el comienzo y el final de una transición son suaves, como se muestra en (c). Para evitar dificultades al definir los puntos extremos, los tiempos de ascenso y de caída se miden normalmente en las fronteras de los niveles lógicos válidos como se indica en la figura.

Con la convención de (c), los tiempos de ascenso y caída indican cuánto tiempo tarda un voltaje de salida en pasar a través de la región indefinida entre BAJO y ALTO. La parte inicial de una transición no se incluye en el valor del tiempo de ascenso o de caída. En vez de ello, la parte inicial de una transición contribuye al valor del "retardo de propagación" que se analiza en la subsección siguiente.

Los tiempos de ascenso y caída de una salida de CMOS dependen principalmente de dos factores, la resistencia del transistor "encendido" y la capacitancia de carga. Una capacitancia grande incrementa los tiempos de transición; como esto es indeseable, es muy raro que un diseñador lógico conecte a propósito un capacitor en la salida del circuito lógico. Sin embargo, la capacitancia parásita está presente en cualquier circuito y proviene de al menos tres fuentes:

capacitancia parásita

1. Circuitos de salida, incluyendo transistores de salida de la compuerta, el alambrado interno y encapsulado, tienen cierta capacitancia asociada a ellos. Esta capacitancia tiene un valor de 2 a 10 picofarads (pF) para las familias lógicas típicas, incluyendo la CMOS.

2. El alambrado que conecta una salida a otras entradas tiene capacitancia, aproximadamente 1 pF por pulgada o más, dependiendo de la tecnología de alambrado.
3. Los circuitos de entrada, incluyendo los transistores, el alambrado interno y encapsulado, tienen una capacitancia de 2 a 15 pF por entrada en familias lógicas típicas.

La capacitancia parásita se conoce en ocasiones como una *carga capacitiva* o *carga de CA*.

carga capacitiva
carga de CA

Se pueden analizar los tiempos de ascenso y caída de una salida CMOS utilizando el circuito equivalente que se indica en la figura 3-37. Como en la sección anterior, los transistores de canal-*p* y canal-*n* están modelados por las resistencias R_p y R_n, respectivamente. En operación normal, una resistencia es alta y la otra es baja, dependiendo del estado de la salida. La carga de la salida se modela mediante un *circuito de carga equivalente* con tres componentes:

circuito de carga equivalente

R_L, V_L Ambos componentes representan la carga de CD: determinan los voltajes y las corrientes que están presentes cuando la salida se estabiliza en un nivel ALTO o BAJO. La carga de CD no tiene demasiado efecto sobre los tiempos de transición cuando la salida cambia de estado.

C_L Esta capacitancia representa la carga de CA: determina los voltajes y corrientes que están presentes cuando cambia la salida, y cuánto tiempo le toma cambiar de un estado a otro.

Cuando una salida CMOS controla solamente entradas CMOS, la carga de CD es despreciable. Para simplificar las cosas, analizaremos solamente este caso, con $R_L = \infty$ y $V_L = 0$, en el resto de esta subsección. La presencia de una carga de CD no despreciable afectaría los resultados, aunque no dramáticamente (véase el ejercicio 3.68).

Ahora podemos analizar los tiempos de transición de una salida CMOS. Para el propósito de este análisis, suponemos que $C_L = 100$ pF, que equivale a una carga capacitiva moderada. También, asumiremos que las resistencias "de encendido" de los transistores de canal-*p* y canal-*n* son 200 Ω y de 100 Ω, respectivamente, como en la subsección anterior. Los tiempos de ascenso y de caída dependen del tiempo que se necesita para cargar y descargar la carga capacitiva C_L.

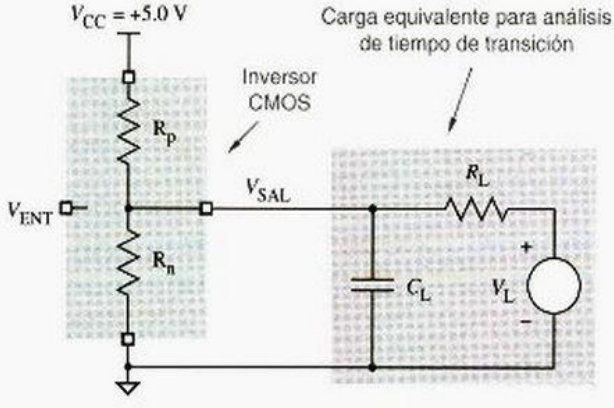


Figura 3-37
Circuito equivalente para analizar los tiempos de transición de una salida CMOS.

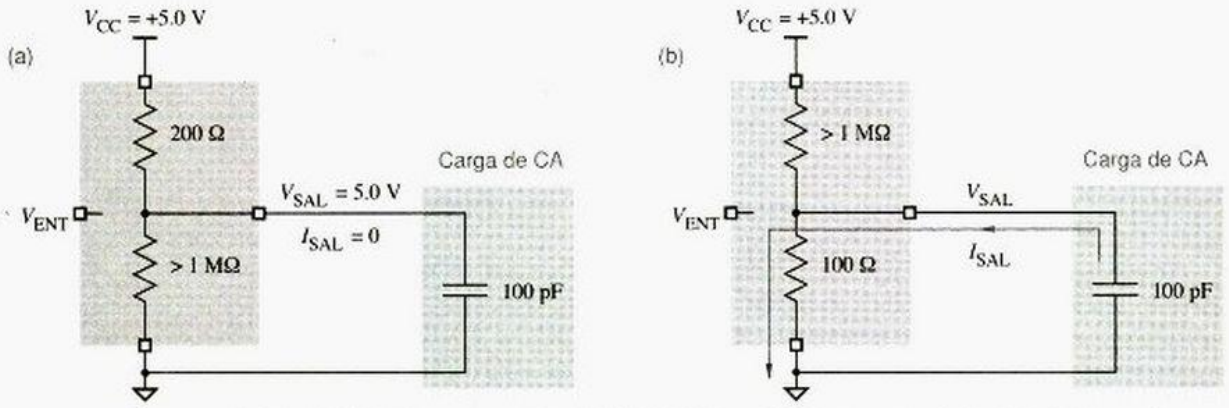


Figura 3-38 Modelo de una transición CMOS ALTO a BAJO: a) en el estado ALTO; b) después de que el transistor de canal-*p* se apaga y el transistor de canal-*n* se enciende.

Primero, observamos el tiempo de caída. La figura 3-38(a) muestra las condiciones eléctricas en el circuito cuando la salida se encuentra en un estado estable ALTO. (No se incluyen R_L y V_L porque no afectan el circuito, ya que $R_L = \infty$.) Para los propósitos de nuestro análisis, podemos suponer lo siguiente: cuando los transistores CMOS cambian entre "encendido" y "apagado", lo hacen en forma instantánea. Supondremos también que al tiempo $t = 0$, la salida CMOS cambia al estado BAJO, generando así la situación que se describe en (b).

Cuando $t = 0$, V_{SAL} todavía es 5.0 V. (Una máxima útil en la ingeniería eléctrica es que el voltaje que circula a través de un capacitor no puede cambiar en forma instantánea.) En el tiempo $t = \infty$, el capacitor debe estar completamente descargado y V_{SAL} debe ser 0 V. Entre los dos momentos, el valor de V_{SAL} está gobernado por una ley exponencial:

$$\begin{aligned} V_{SAL} &= V_{DD} \cdot e^{-t/(R_n C_L)} \\ &= 5.0 \cdot e^{-t/(100 \cdot 100 \cdot 10^{-12})} \\ &= 5.0 \cdot e^{-t/(10 \cdot 10^{-9})} \text{ V} \end{aligned}$$

constante de tiempo RC

El factor $R_n C_L$ tiene unidades de tiempo (segundos) y se conoce como *constante de tiempo RC*. El cálculo que sigue muestra que la constante de tiempo *RC* para las transiciones ALTO a BAJO es de 10 nanosegundos (ns).

La figura 3-39 muestra una gráfica de V_{SAL} en función de tiempo. Para calcular el tiempo de caída, recuerde que 1.5 V y 3.5 V son las fronteras que definimos para los niveles BAJO y ALTO de las entradas CMOS, que están controladas por la salida CMOS. Para obtener el tiempo de caída, debemos resolver la ecuación anterior para $V_{SAL} = 3.5$ y $V_{SAL} = 1.5$, para obtener:

$$t = -R_n C_L \cdot \ln \frac{V_{SAL}}{V_{DD}} = -10 \cdot 10^{-9} \cdot \ln \frac{V_{SAL}}{5.0}$$

$$t_{3.5} = 3.57 \text{ ns}$$

$$t_{1.5} = 12.04 \text{ ns}$$

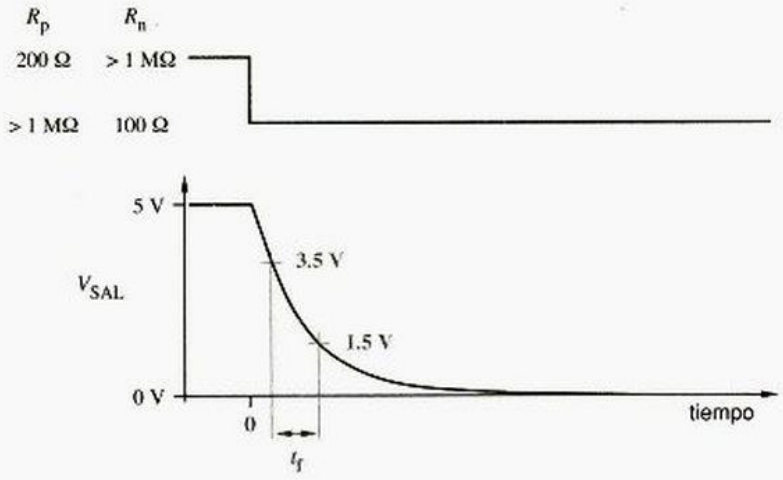


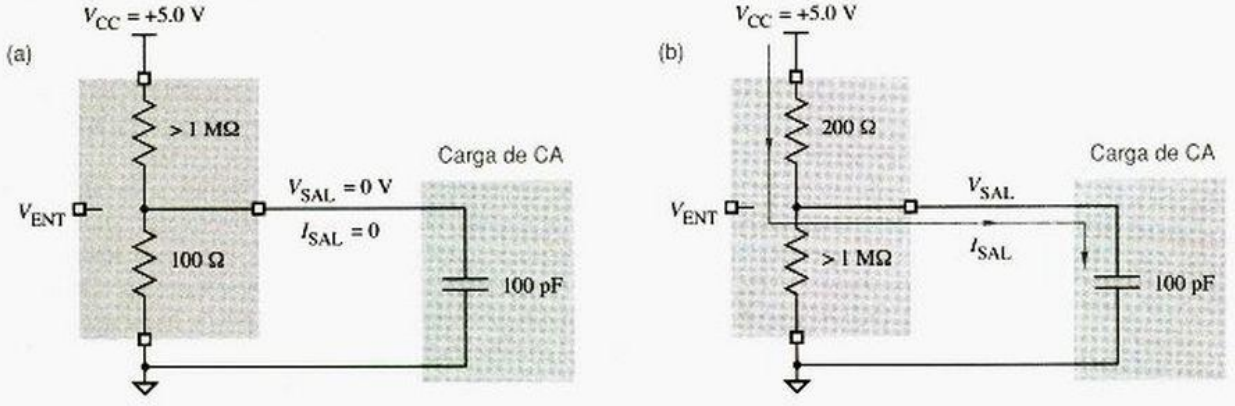
Figura 3-39
Tiempo de caída para una transición de ALTO a BAJO de una salida CMOS.

El tiempo de caída t_f es la diferencia entre ambos números, o aproximadamente 8.5 ns.

El tiempo de ascenso puede calcularse con un método semejante. La figura 3-40(a) muestra las condiciones en el circuito cuando la salida se encuentra en un estado estable BAJO. Si al tiempo $t = 0$ la salida CMOS cambia al estado ALTO, resulta la situación descrita en (b). Una vez más, V_{SAL} no puede cambiar de forma instantánea, pero al tiempo $t = \infty$, el capacitor estará completamente cargado y V_{SAL} será 5.0 V. De nueva cuenta, el valor de V_{SAL} entre los dos está determinado por una ley exponencial:

$$\begin{aligned} V_{SAL} &= V_{DD} \cdot (1 - e^{-t/(R_p C_L)}) \\ &= 5.0 \cdot (1 - e^{-t(200 \cdot 100 \cdot 10^{-12})}) \\ &= 5.0 \cdot (1 - e^{-t/(20 \cdot 10^{-9})}) \text{ V} \end{aligned}$$

Figura 3-40 Modelo de una transición CMOS de BAJO a ALTO: a) en el estado BAJO; b) después de que el transistor de canal-*n* se apaga y el transistor de canal-*p* se enciende.



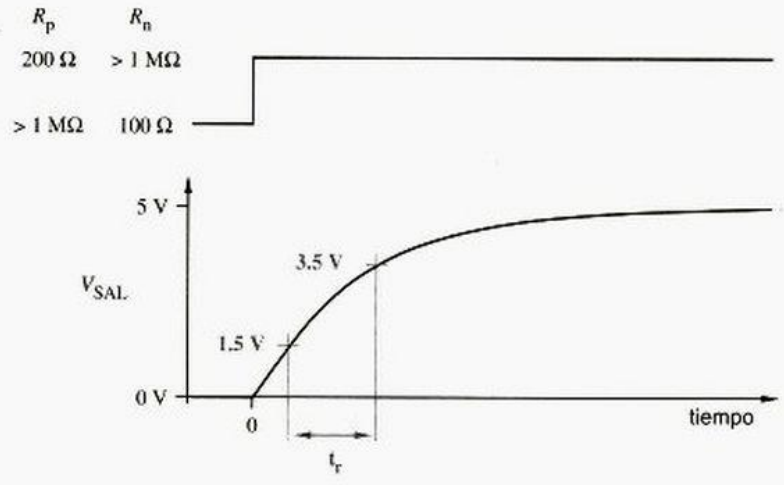


Figura 3-41
Tiempo de ascenso para una transición de BAJO a ALTO de una salida CMOS.

En este caso la constante de tiempo RC es 20 ns. La figura 3-41 muestra una gráfica de V_{SAL} en función del tiempo. Para obtener el tiempo de ascenso, debemos resolver la ecuación anterior para $V_{SAL} = 1.5$ y $V_{SAL} = 3.5$, para obtener

$$t = -RC \cdot \ln \frac{V_{DD} - V_{SAL}}{V_{DD}}$$

$$= -20 \cdot 10^{-9} \cdot \ln \frac{5.0 - V_{SAL}}{5.0}$$

$$t_{1.5} = 7.13 \text{ ns}$$

$$t_{3.5} = 24.08 \text{ ns}$$

El tiempo de ascenso t_r es la diferencia entre estos dos números, o aproximadamente 17 ns.

El ejemplo anterior supone que el transistor de canal-p tiene dos veces la resistencia del transistor de canal-n, y como resultado el tiempo de ascenso es el doble de extenso que el tiempo de caída. El "débil" transistor de canal-p necesita más tiempo para elevar la salida en comparación con el "fuerte" transistor de canal-n que requiere menos tiempo para disminuirla; la capacidad de excitación de la salida es "asimétrica". En ocasiones los dispositivos CMOS de alta velocidad se fabrican con transistores de canal-p de mayor tamaño para igualar los tiempos de transición y obtener una excitación de salida más simétrica.

Sin importar las características de los transistores, un incremento en la capacitancia de carga ocasionará un aumento en la constante de tiempo RC y el correspondiente incremento en los tiempos de transición de la salida. Por lo tanto, la meta de los diseñadores de circuitos de alta velocidad es minimizar la capacitancia de carga, especialmente en la mayor parte de las señales críticas de temporización. Esto puede hacerse minimizando la cantidad de entradas que excita la señal, creando múltiples copias de la señal y mediante una cuidadosa ubicación de los componentes del circuito (layout).

Cuando se trabaja con circuitos digitales reales, resulta útil estimar los tiempos de transición sin tener que realizar un análisis detallado. Una regla sencilla y práctica es que el tiempo de transición es aproximadamente igual a la constante de tiempo *RC* del circuito, cuando se está cargando o descargando. Por ejemplo, el valor estimado de 10 o 20 ns para los tiempos de ascenso y caída del ejemplo anterior es más adecuado, si consideramos que la mayoría de las suposiciones acerca de la capacitancia de carga y las resistencias de "encendido" de los transistores son aproximaciones iniciales de nuestro análisis.

En general, los fabricantes de circuitos comerciales CMOS no especifican las resistencias de "encendido" del transistor en sus hojas de datos. Puede encontrar esta información en las hojas de aplicación de los fabricantes. En cualquier caso, puede estimar el valor de la resistencia de "encendido" como la caída de voltaje en el transistor "encendido" dividida entre la corriente que circula a través de él, considerando la carga resistiva en el peor de los casos, como se indica en la sección 3.5.2:

$$R_{\text{p(encendido)}} = \frac{V_{\text{DD}} - V_{\text{OHminT}}}{|I_{\text{OHmaxT}}|}$$

$$R_{\text{n(encendido)}} = \frac{V_{\text{OLmaxT}}}{|I_{\text{OLmaxT}}|}$$

¡HAY UNA TRAMPA! Los tiempos de transición calculados son en realidad bastante sensibles a la selección de los niveles lógicos. En los ejemplos de esta subsección, si utilizamos 2.0 V y 3.0 V en vez de 1.5 V y 3.5 V como los umbrales para BAJO y ALTO, calcularíamos tiempos de transición más cortos. Por otro lado, si empleáramos 0.0 y 5.0 V, ¡los tiempos de transición calculados serían infinitos! También debería tener cuidado porque en algunas familias lógicas (más notablemente en TTL), los umbrales no son simétricos en torno del punto medio de voltaje. A pesar de todo, en la experiencia del autor la regla práctica "constante de tiempo igual a tiempo de transición" por lo regular funciona para circuitos prácticos.

3.6.2 Retardo de propagación

Los tiempos de ascenso y caída solamente describen en forma parcial el comportamiento dinámico de un elemento lógico; se necesitan parámetros adicionales para relacionar la temporización de salida con la temporización de entrada. Una *trayectoria de señal* es la ruta eléctrica de una señal particular de entrada hacia una señal particular de salida, en un elemento lógico. El *retardo de propagación* t_p de una trayectoria de señal es la cantidad de tiempo que requiere la señal de entrada para producir un cambio en la señal de salida.

trayectoria de señal

retardo de propagación t_p

Un elemento lógico complejo con varias entradas y salidas puede especificar un valor diferente de t_p para cada trayectoria diferente de señal. Además, se pueden especificar valores diferentes para una trayectoria de señal en particular, dependiendo de la dirección de cambio en la salida. Sin tomar en cuenta los tiempos de ascenso y caída, la figura 3-42(a) muestra dos diferentes retardos de propagación para la trayectoria de

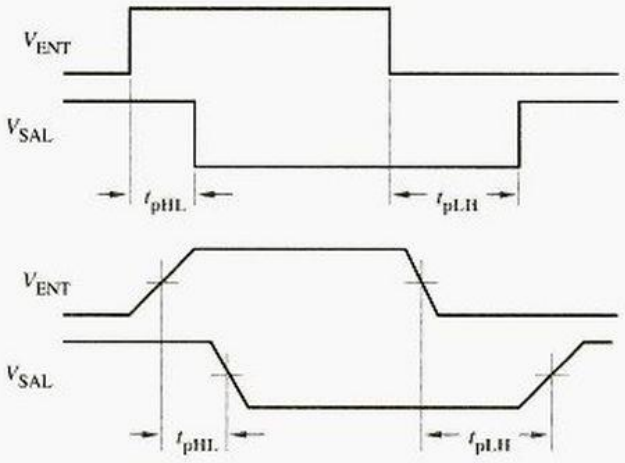


Figura 3-42
Retardos de propagación para un inversor CMOS: a) ignorando tiempos de ascenso y caída; b) medidos en los puntos medios de las transiciones.

señal de entrada a salida de un inversor CMOS, dependiendo de la dirección del cambio en la salida:

t_{pHL}

t_{pHL} El tiempo entre un cambio de entrada y el correspondiente cambio de salida, cuando la salida está cambiando de ALTO a BAJO.

t_{pLH}

t_{pLH} El tiempo entre un cambio de entrada y el correspondiente cambio de salida, cuando la salida está cambiando de BAJO a ALTO.

Varios factores generan retardos de propagación diferentes de cero. En un dispositivo CMOS, la velocidad a la cual los transistores cambian de estado está influenciada tanto por la física de los semiconductores del dispositivo como por el entorno del circuito —que incluye la velocidad de transición de la señal de entrada, la capacitancia de entrada y carga de salida—. En los dispositivos que tienen etapas múltiples (p. ej. compuertas no inversoras o funciones lógicas más complejas) puede ser necesario que varios transistores internos cambien de estado para que la salida pueda a su vez cambiar de estado. Incluso cuando la salida comienza a cambiar de estado, con tiempos de ascenso y caída diferentes de cero, se necesita cierta cantidad de tiempo para cruzar la región entre ambos estados, como se indica en la la subsección anterior. Todos estos factores están incluidos en el retardo de propagación.

Para enfatizar el efecto de los tiempos de ascenso y caída, es normal que los fabricantes especifiquen los tiempos de retardo de propagación como los puntos medios de las transiciones de entrada y salida, como se ilustra en la figura 3-42(b). Sin embargo, en ocasiones los retardos se especifican en los puntos de frontera del nivel lógico, en especial si los tiempos de ascenso y caída son "lentos" pueden afectar la operación del dispositivo. Por ejemplo, la figura 3-43 muestra cómo podría especificarse el ancho mínimo del pulso de entrada de un latch SR, el cual se analiza en la sección 7.2.1.

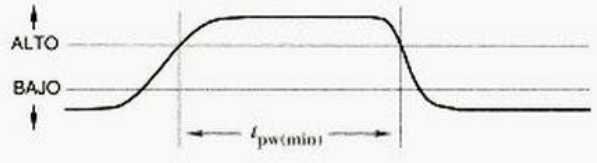


Figura 3-43
Temporización en el peor de los casos especificada usando puntos de frontera de nivel lógico.

Además, un fabricante puede especificar los tiempos de ascenso y caída de entrada máxima absoluta que deben satisfacerse para garantizar un funcionamiento apropiado. Los circuitos CMOS de alta velocidad pueden consumir corriente excesiva u oscilar si sus transiciones de entrada son demasiado lentas.

3.6.3 Consumo de energía

disipación de energía en estado de reposo

disipación estática de energía

El consumo de energía de un circuito CMOS cuya salida no cambia se llama *disipación estática de energía* o *disipación de energía en reposo*. (Las palabras *consumo* y *disipación* se utilizan en forma indistinta cuando se analiza la cantidad de energía que utiliza un dispositivo.) La mayor parte de los circuitos CMOS tienen poca disipación estática de energía. Esto es lo que los hace tan atractivos para las computadoras portátiles y otras aplicaciones de baja potencia (cuando no se realizan cálculos se consume poca energía). Un circuito CMOS consume energía en forma significativa durante las transiciones; esto se conoce como *disipación dinámica de energía*.

disipación dinámica de energía

Una fuente de disipación dinámica de energía es un cortocircuito parcial en la estructura de salida del dispositivo CMOS. Cuando el voltaje de entrada no es cercano al voltaje de alguna de las líneas de alimentación (0 V o V_{CC}), tanto los transistores de salida de canal-p como los de canal-n pueden estar "encendidos" en forma parcial; esto puede generar una resistencia en serie igual o inferior a 600 Ω. En este caso, la corriente fluye a través de los transistores desde V_{CC} hasta tierra. La cantidad de energía que se consume de esta forma depende tanto del valor de V_{CC} como de la velocidad a la que ocurren las transiciones de salida, de acuerdo con la fórmula

$$P_T = C_{PD} \cdot V_{CC}^2 \cdot f$$

En la fórmula, se utilizan las siguientes variables:

- P_T La disipación de energía interna del circuito debido a las transiciones de salida.
- V_{CC} El voltaje de la fuente de alimentación. Como todos los ingenieros eléctricos lo saben, la disipación de energía a través de una carga resistiva (los transistores parcialmente encendidos) es proporcional al *cuadrado* del voltaje.
- f La *frecuencia de transición* de la señal de salida. Ésta especifica el número de transiciones de salida —que consumen energía— por segundo. (Pero note que la frecuencia se define como el número de transiciones dividida entre 2.)

frecuencia de transición

- C_{PD} La *capacitancia de disipación de energía*. Esta constante, la especifica normalmente el fabricante del dispositivo, completa la fórmula. C_{PD} resulta tener unidades de capacitancia, pero no representa la capacitancia real de salida. En vez de ello, incorpora la dinámica del flujo de corriente a través de las resistencias cambiantes del transistor de salida durante un par sencillo de transiciones de salida, ALTO a BAJO y BAJO a ALTO. Por ejemplo, el valor de C_{PD} para compuertas CMOS de la serie HC es 20-24 pF, aun cuando la capacitancia de salida real sea mucho menor.

capacitancia de disipación de energía

La fórmula P_T es válida solamente si las transiciones de entrada tienen la rapidez suficiente para producir transiciones veloces en la salida. Si las transiciones de entrada son demasiado lentas, entonces los transistores de salida permanecen parcialmente

encendidos durante más tiempo, y se incrementa el consumo de energía. En general, los fabricantes de dispositivos recomiendan que el tiempo máximo de ascenso y caída de entrada tenga un valor inferior al que se especifica para C_{PD}.

C_L

La segunda, y con frecuencia la fuente de consumo de energía más importante de un dispositivo CMOS, es la carga capacitiva (C_L) de la salida. Durante una transición BAJO a ALTO, la corriente fluye a través de un transistor de canal-*p* para cargar a C_L. Del mismo modo, durante una transición ALTO a BAJO, la corriente fluye a través de un transistor de canal-*n* para descargar a C_L. En cada caso, se disipa energía en la resistencia "de encendido" del transistor. Utilizaremos P_L para indicar la cantidad total de energía que se disipa cuando se carga y descarga C_L.

P_L

Las unidades de P_L son de energía, o uso de energía por unidad de tiempo. Se puede determinar la energía de una transición al calcular la corriente que circula a través del transistor de carga como una función del tiempo (utilizando la constante de tiempo *RC* como en la sección 3.6.1), elevando al cuadrado esta función, multiplicándola por la resistencia "de encendido" del transistor de carga, e integrando sobre el tiempo. A continuación se describe una forma más fácil.

Durante una transición, el voltaje a través de la capacitancia de carga C_L cambia por $\pm V_{CC}$. De acuerdo con la definición de capacitancia, la cantidad total de carga que debe fluir para producir un cambio de voltaje en V_{CC} a través de C_L es $C_L \cdot V_{CC}$. La cantidad total de energía que se utiliza en una transición es la carga por el cambio en el voltaje promedio. El primer segmento de carga genera un cambio de voltaje en V_{CC}, mientras que el último segmento de carga produce un cambio pequeño e imperceptible en el voltaje, de aquí que el cambio promedio es $V_{CC}/2$. Por consiguiente, la energía total por transición es $C_L \cdot V_{CC}^2/2$. Si tenemos $2f$ transiciones por segundo, la energía total disipada debido a la carga capacitiva es

$$\begin{aligned} P_L &= C_L \cdot (V_{CC}^2/2) \cdot 2f \\ &= C_L \cdot V_{CC}^2 \cdot f \end{aligned}$$

La disipación dinámica total de energía de un circuito CMOS es la suma de P_T y P_L:

$$\begin{aligned} P_D &= P_T + P_L \\ &= C_{PD} \cdot V_{CC}^2 \cdot f + C_L \cdot V_{CC}^2 \cdot f \\ &= (C_{PD} + C_L) \cdot V_{CC}^2 \cdot f \end{aligned}$$

energía CV^2f

Basados en esta fórmula, la disipación dinámica de energía se conoce también como *energía CV^2f*. En la mayoría de las aplicaciones de los circuitos CMOS, la energía CV^2f es la que contribuye en mayor parte a la disipación total de energía. Cabe indicar que la energía CV^2f también se consume en los circuitos lógicos bipolares como TTL y ECL, pero a frecuencias bajas o moderadas es insignificante en comparación con la disipación estática de energía (de CD o en reposo) de los circuitos bipolares.

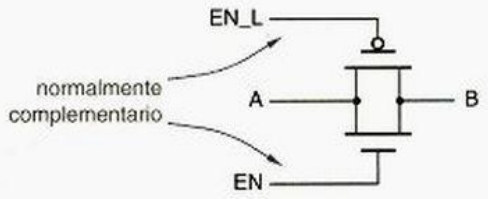


Figura 3-44
Compuerta de transmisión CMOS.

3.7 Otras estructuras CMOS de entrada y salida

Los diseñadores de circuitos han modificado el circuito CMOS básico en muchas maneras para producir compuertas que se ajustan a aplicaciones específicas. Esta sección describe algunas de las variaciones más comunes en las estructuras de entrada y salida CMOS.

3.7.1 Compuertas de transmisión

Un par de transistores de canal-*p* y de canal-*n* pueden conectarse entre sí para formar un interruptor controlado por lógica. La figura 3-44(a) muestra este circuito, que también se conoce con el nombre de *compuerta de transmisión* CMOS.

compuerta de transmisión

Una compuerta de transmisión se hace funcionar de modo que sus señales de entrada EN y EN_L siempre se encuentren en niveles opuestos. Cuando EN es ALTO y EN_L es BAJO, existe una conexión de baja impedancia (tan baja como 2-5 Ω) entre los puntos A y B. Cuando EN es BAJO y EN_L es ALTO, los puntos A y B están desconectados.

Una vez que una compuerta de transmisión se encuentra habilitada, el retardo de propagación de A hacia B (o viceversa) es muy breve. Debido a sus retardos cortos y simplicidad conceptual, las compuertas de transmisión se incluyen el interior de los dispositivos CMOS de gran escala (como multiplicadores y flip-flops). Por ejemplo, la figura 3-45 muestra el uso de las compuertas de transmisión para crear un "multiplexor de 2 entradas". Cuando S es BAJO, la "entrada" X se conecta a la "salida" Z; cuando S es ALTO, Y se conecta a Z.

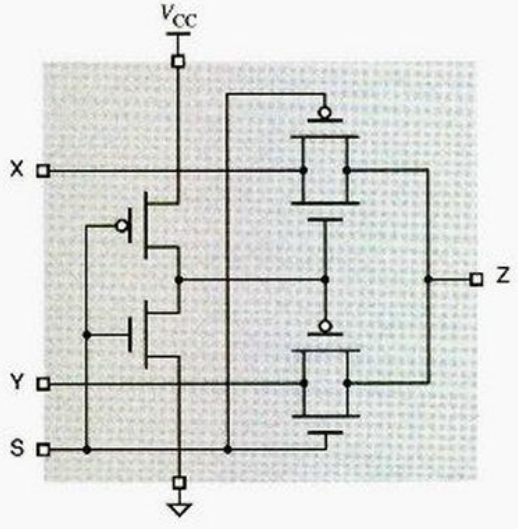


Figura 3-45
Multiplexor de dos entradas utilizando compuertas de transmisión CMOS.

Al menos un fabricante comercial (Integrated Device Technology) ofrece una amplia variedad de funciones lógicas que se basan en compuertas de transmisión. En sus dispositivos multiplexores, un cambio en las entradas "seleccionadas" requiere varios nanosegundos (tal como en la figura 3-45) para afectar la trayectoria de entrada-salida (X o Y hacia Z). Sin embargo, una vez que se ha establecido la trayectoria, el retardo de propagación de la entrada a la salida está especificado como máximo en 0.25 ns; éste es el multiplexor discreto CMOS más rápido que usted podrá adquirir.

3.7.2 Entradas de un disparador Schmitt

entrada de disparador Schmitt

La curva característica de transferencia de entrada-salida para una compuerta típica CMOS se mostró en la figura 3-25 de la página 100. La figura 3-46(a) muestra la curva característica de transferencia de una compuerta con *entradas de disparador Schmitt*. Un disparador Schmitt es un circuito especial que utiliza retroalimentación interna para desplazar el umbral de conmutación dependiendo de si la entrada está cambiando de BAJO a ALTO o de ALTO a BAJO.

Por ejemplo, supongamos que la entrada de un inversor con disparador Schmitt se encuentra inicialmente en 0 V, un nivel BAJO estable. Entonces la salida es ALTO, cercana a 5.0 V. Si el voltaje de entrada se incrementa, la salida no irá a BAJO hasta que el voltaje de entrada alcance aproximadamente 2.9 V. Sin embargo, una vez que la salida cambia a BAJO, no irá a ALTO hasta que la entrada disminuya en aproximadamente 2.1 V. De este modo, el umbral de conmutación para los cambios de entrada que tienen excursión positiva, indicado como V_{T+}, es aproximadamente de 2.9 V. Para los cambios de entrada que tienen excursión negativa, el umbral de conmutación (V_{T-}) es aproximadamente 2.1 V. La diferencia entre ambos umbrales se denomina *histéresis*. El inversor con disparador Schmitt proporciona casi 0.8 V de histéresis.

histéresis

Para demostrar la utilidad de la histéresis, la figura 3-47(a) muestra una señal de entrada con tiempos de ascenso y caída largos, y aproximadamente 0.5 V de ruido. Un inversor sencillo, sin histéresis, presenta el mismo umbral de conmutación tanto para las transiciones que tienen tendencia (excursión) positiva como negativa. $V_T \approx 2.5$ V. Así, el inversor simple responde al ruido como se muestra en (b), y produce varios cambios en la salida cada vez que el voltaje de entrada —con ruido— cruza el umbral de conmutación. No obstante, como se muestra en (c), un inversor con disparador Schmitt no responde al ruido, porque su histéresis es mayor que la amplitud del ruido.

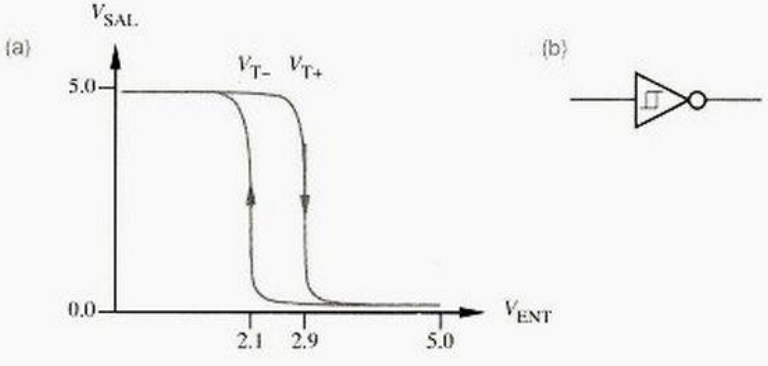


Figura 3-46
inversor de disparador Schmitt: a) característica de transferencia entrada-salida; b) símbolo lógico.

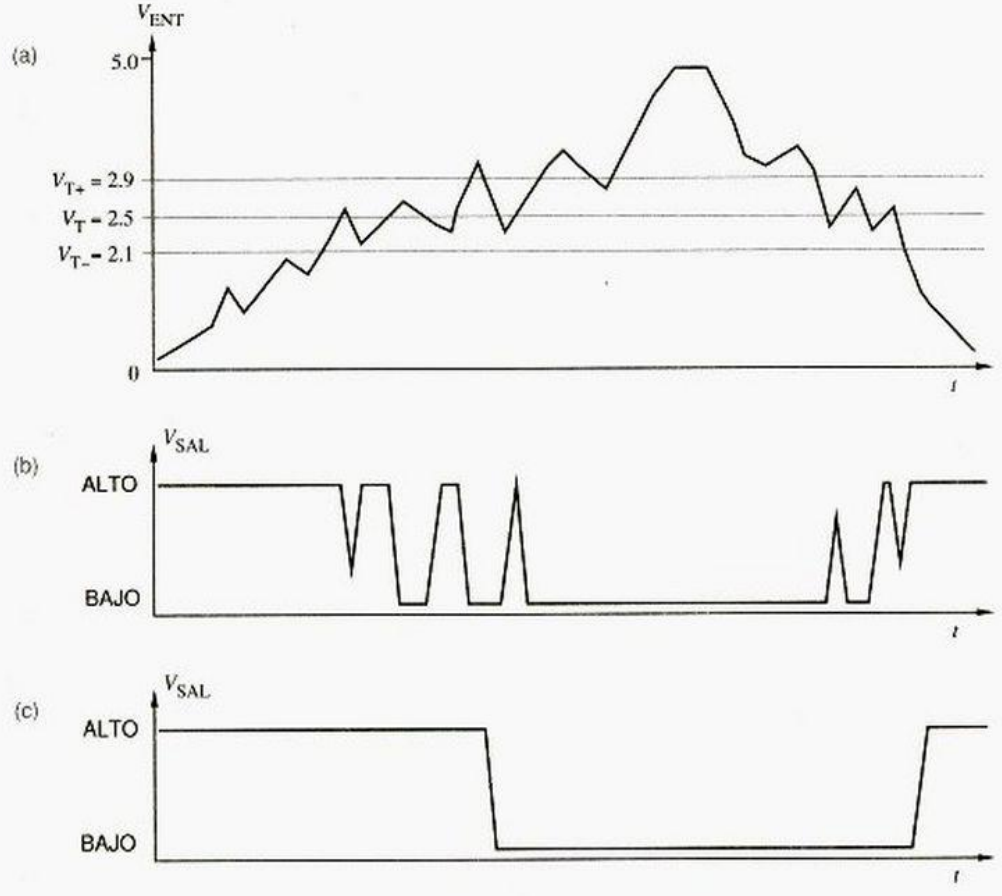
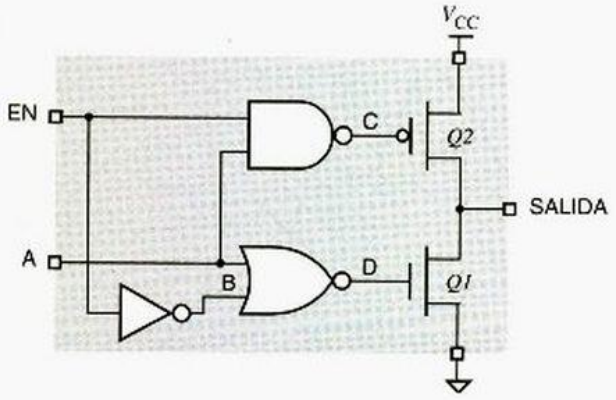


Figura 3-47 Operación del dispositivo con entradas que cambian lentamente: a) una entrada que cambia lentamente con ruido; b) salida producida por un inversor ordinario; c) salida producida por un inversor con 0.8 V de histéresis.

ARREGO DE SU TRANSMISIÓN

Las entradas de disparador Schmitt tienen mejor inmunidad al ruido que las entradas de compuerta ordinarias para señales que contienen reflejos de línea de transmisión, estudiadas en la sección 11.4, o que tienen tiempos largos de ascenso y de caída. Tales señales se presentan por lo regular en conexiones físicamente largas, tales como buses de entrada-salida y cables de interfaz de computadoras. La inmunidad al ruido es importante en estas aplicaciones, puesto que las líneas de señal largas tienen más probabilidad de tener reflejos o de captar ruido de líneas de señal, circuitos y aparatos adyacentes.



(b)

EN	A	B	C	D	Q1	Q2	SALIDA
L	L	H	H	L	off	off	Hi-Z
L	H	H	H	L	off	off	Hi-Z
H	L	L	H	H	on	off	L
H	H	L	L	L	off	on	H

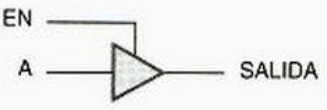


Figura 3-48 Buffer de tres estados CMOS: a) diagrama de circuito; b) tabla de funciones; c) símbolo lógico.

3.7.3 Salidas de tres estados

estado de alta impedancia
estado Hi-Z
estado de flotación

Las salidas lógicas tienen dos estados normales, ALTO y BAJO, que corresponden a los valores lógicos 0 y 1. Sin embargo, algunas salidas tienen un tercer estado eléctrico que no es un estado lógico en absoluto, el cual se denomina *estado de alta impedancia, Hi-Z,* o *de flotación*. En este estado, la salida se comporta como si aún no estuviera conectada al circuito, excepto por una pequeña corriente de fuga que puede fluir hacia dentro o hacia fuera de la terminal de salida. De esta forma, una salida puede tener uno de tres estados: 0 lógico, 1 lógico y de alta impedancia.

salida de tres estados
salida triestado

Una salida con tres estados posibles se conoce como (¡sorpresa!) *salida de tres estados* o, en ocasiones, *salida triestado*. Los dispositivos de tres estados tienen una entrada extra, que por lo regular se llama "habilitación de salida" o "deshabilitación de salida", para colocar la(s) salida(s) del dispositivo en el estado de alta impedancia.

bus de tres estados

Un *bus de tres estados* se produce al conectar entre sí varias salidas de triple estado. Los circuitos de control para las "habilitaciones de salida" deben asegurar que, como máximo, se habilite una salida (pero no en su estado Hi-Z) en cualquier momento. El dispositivo con una sola habilitación puede transmitir niveles lógicos (ALTO y BAJO) en el bus. La sección 5.6 muestra ejemplos de diseño del bus de tres estados.

buffer de tres estados

La figura 3-48(a) muestra el diagrama del circuito de un *buffer CMOS de tres estados*. Para simplificar el diagrama, las funciones internas NAND, NOR e inversora se muestran en forma funcional en vez de su forma transistorizada; en realidad se utiliza un total de 10 transistores (véase el ejercicio 3.80). Como se ilustra en la tabla de funciones (b), cuando la entrada de habilitación (EN, "enable") está en nivel BAJO, ambos transistores de salida están apagados, y la salida está en el estado de alta impedancia. De otro modo, la salida es ALTO o BAJO como se controla mediante la entrada de "datos" A.

NOTICIA LEGAL "TRI-STATE" es una marca registrada de National Semiconductor Corporation. Sus abogados pensaron que a usted le gustaría saberlo.

Los símbolos lógicos para los buffers y las compuertas de tres estados se representan normalmente con la entrada de habilitación en la parte superior, como se ilustra en (c).

En la práctica, el circuito de control de tres estados puede ser diferente al que hemos mostrado, a fin de proporcionar el comportamiento dinámico adecuado del transistor de salida durante las transiciones hacia y desde el estado de alta impedancia. En especial, los dispositivos con salidas de tres estados se diseñan normalmente de modo que el retardo de habilitación de salida (de Hi-Z a BAJO o ALTO) sea algo más largo que el retardo de deshabilitación de salida (BAJO o ALTO hacia Hi-Z). De este modo, si un circuito de control activa la entrada de habilitación de salida del primer dispositivo al mismo tiempo que desactiva la entrada de un segundo dispositivo, se garantiza que el segundo dispositivo entrará al estado de alta impedancia, antes de que el primero coloque un nivel ALTO o BAJO en el bus.

Si ambas salidas de tres estados (en el mismo bus) se habilitan al mismo tiempo e intentan mantener estados opuestos, la situación es similar a la que se obtiene cuando se conectan entre sí las salidas estándar de pull-up activo, como se indica en la figura 3-56 de la página 133: se produce un voltaje no lógico en el bus. Si el conflicto es momentáneo los dispositivos probablemente no se dañarán, pero cantidad considerable de corriente de drenado que circula a través de las salidas (que están unidas) puede producir pulsos de ruido que afectarán el comportamiento del circuito en algún otro sitio del sistema.

Existe una corriente de fuga hasta de 10 μA, que está asociada con una salida CMOS de triple estado en su estado de alta impedancia. Esta corriente, así como las corrientes de entrada de las compuertas receptoras, debe tomarse en cuenta cuando se calcula el máximo número de dispositivos que se pueden colocar en un bus de tres estados. Es decir, en el estado BAJO o ALTO, una salida habilitada de tres estados debe ser capaz de extraer o suministrar hasta 10 μA de corriente de fuga para cada una de las otras salidas de tres estados en el bus, también debe manejar la corriente que requiere cada entrada del bus. Como sucede en la lógica estándar CMOS, deben hacerse cálculos separados de estado BAJO y estado ALTO para asegurar que se satisfacen los requerimientos de fan out de la configuración particular de un circuito.

*3.7.4 Salidas de drenaje abierto

Se dice que los transistores de canal-*p* en las estructuras de salida CMOS proporcionan *arranque activo*, puesto que aumentan activamente el voltaje de salida en una transición de BAJO a ALTO. Estos transistores se omiten en las compuertas que tienen *salidas de drenaje abierto*, como la compuerta NAND de la figura 3-49(a). El drenaje del transistor de canal-*n* del extremo superior se deja desconectado internamente, por consiguiente si la salida no es BAJO está "abierta", como se indica en (b). El diamante subrayado en la figura (c) se utiliza en ocasiones para indicar una salida de drenaje abierto. Una estructura similar, también conocida como "salida de colector abierto", se proporciona en las familias lógicas TTL como se describe en la sección 3.10.5.

arranque activo (active pull-up)

salida de drenaje abierto

Una salida de drenaje abierto necesita un *resistor externo de arranque (external pull-up resistor)* para proporcionar un *arranque pasivo* (passive pull-up) al nivel ALTO. Por ejemplo, la figura 3-50 muestra una compuerta NAND CMOS con drenaje abierto, en este caso su resistor de arranque controla una carga.

resistor de arranque (pull-up resistor)

arranque pasivo (passive pull-up)

*En este libro, las secciones opcionales están marcadas con un asterisco.

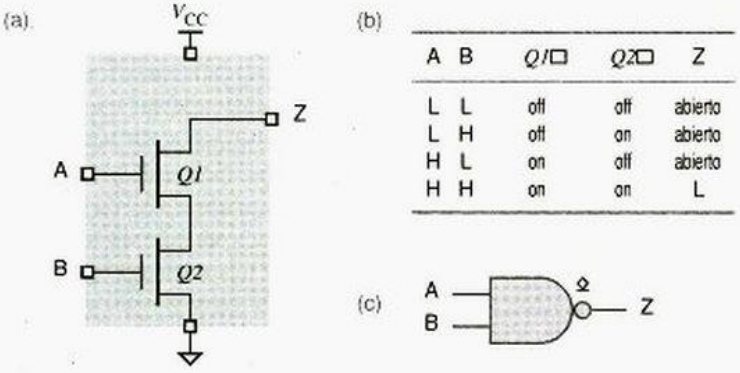


Figura 3-49
Compuerta NAND CMOS de drenaje abierto: a) diagrama del circuito; b) tabla de funciones; c) símbolo lógico.

Para obtener la velocidad más alta posible, el valor del resistor de arranque (con salida de drenaje abierto) debería ser tan pequeño como sea posible; esto minimiza la constante de tiempo *RC* para las transiciones BAJO a ALTO (tiempo de ascenso). Sin embargo, el valor del resistor de arranque no puede ser arbitrariamente pequeño; la resistencia mínima está determinada por la corriente máxima que consumen las salidas de drenaje abierto, I_{OLmax}. Por ejemplo, en los circuitos CMOS serie HC y HCT, I_{OLmax} es 4 mA y el resistor de arranque no puede ser menor a 5.0 V/4 mA, o bien, 1.25 kΩ. Puesto que esto es un orden de magnitud mayor que la resistencia de "encendido" de los transistores de canal-*p* en una compuerta CMOS estándar, las transiciones de salida de BAJO a ALTO son mucho más lentas para una compuerta con drenaje abierto que para una compuerta estándar con arranque activo.

Como un ejemplo, supongamos que la compuerta de drenaje abierto de la figura 3-50 es CMOS serie HC, la resistencia de arranque es 1.5 kΩ y la capacitancia de carga es 100 pF. Anteriormente demostramos en la sección 3.5.2 que la resistencia de "encendido" de una salida CMOS serie HC en el estado BAJO es de aproximadamente 80 Ω. Por consiguiente, la constante de tiempo de una transición ALTO a BAJO es de aproximadamente (80Ω) (100 pF) = 8 ns, y el tiempo de caída de la salida es aproximadamente 8 ns. Sin embargo, la constante de tiempo RC para una transición BAJO a ALTO es de aproximadamente (1.5 kΩ)(100 pF) = 150 ns, y el tiempo de ascenso es 150 ns. Este valor del tiempo de ascenso relativamente lento contrasta con el tiempo de caída mucho más rápido de la figura 3-51. (Un amigo del autor llama *ooze* a este tipo de transiciones de ascenso lentas).

Ooze

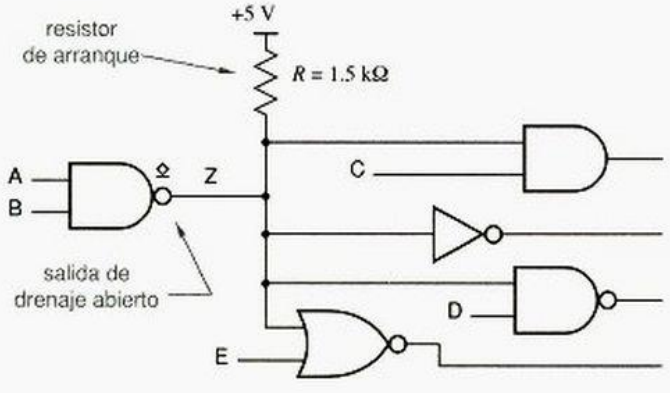


Figura 3-50
Compuerta NAND CMOS de drenaje abierto controlando una carga.

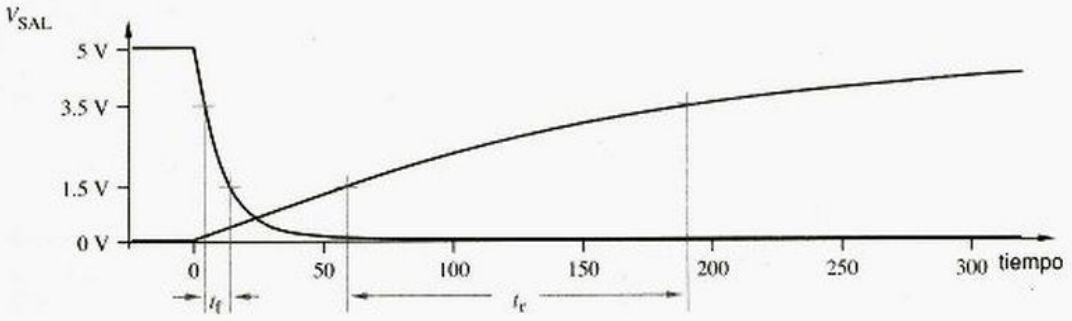


Figura 3-51 Transiciones de ascenso y caída de una salida CMOS de drenaje abierto.

¿Por qué utilizar salidas de drenaje abierto? A pesar de los tiempos de ascenso lentos, pueden ser útiles en por lo menos tres aplicaciones: control de diodos emisores de luz (los LED, del inglés light-emitting diodes) y otros dispositivos; el desarrollo de lógica alambrada y el control de buses de fuente múltiple.

*3.7.5 Control de LED

Una salida del tipo drenaje abierto puede excitar a un LED como se ilustra en la figura 3-52. Si cualquiera de las entradas A o B es BAJO, el correspondiente transistor de canal-*n* estará apagado y el LED no se encenderá. Cuando ambas entradas A y B sean ALTO, los dos transistores estarán encendidos, la salida Z estará en nivel BAJO y el LED se encenderá. El valor del resistor de arranque *R* se selecciona para que fluya una cantidad apropiada de corriente a través del LED en el estado "encendido".

Los LED típicos necesitan 10 mA para que "brillen" en forma normal. Las salidas CMOS de las series HC y HCT están especificadas solamente para consumir 4 mA y normalmente no se utilizan para controlar LEDs. Sin embargo, las salidas de familias CMOS avanzadas tales como la 74AC y 74ACT pueden consumir 24 mA o más, por lo tanto se pueden utilizar para controlar los LED con bastante eficacia.

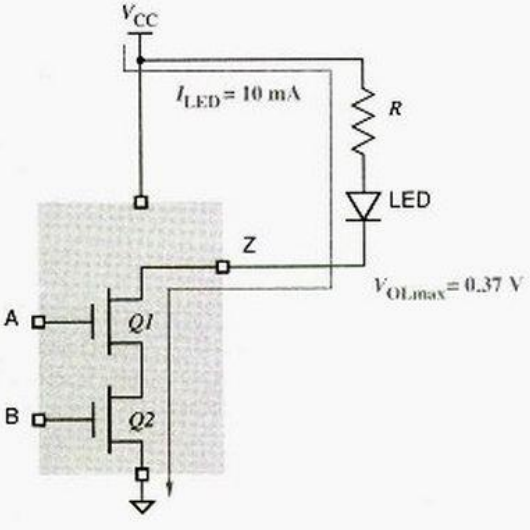


Figura 3-52
Control de un LED con una salida de drenaje abierto.

Se necesitan tres segmentos de información para calcular el valor apropiado del resistor de arranque R:

1. La corriente que se necesita para obtener el brillo deseado en el LED I_{LED}, 10 mA para los LED típicos.
2. La caída de voltaje V_{LED} en el LED en la condición "encendido", aproximadamente 1.6 V para los LED típicos.
3. El voltaje de salida V_{OL} de la salida de drenaje abierto que consume la corriente del LED. En las familias CMOS 74AC y 74ACT, V_{OLmax} es 0.37 V. Si una salida puede consumir una corriente I_{LED} y mantener un voltaje más bajo, digamos, 0.2 V, entonces el cálculo que se muestra a continuación indicará el uso de un resistor cuyo valor es bastante pequeño, aunque esto no producirá ningún daño. Circulará una cantidad ligeramente mayor que I_{LED} y el brillo del LED será ligeramente superior a lo esperado.

Utilizando la información anterior, podemos escribir la ecuación siguiente:

$$V_{OL} + V_{LED} + (I_{LED} \cdot R) = V_{CC}$$

Suponiendo que $V_{CC} = 5.0$ V y los valores típicos en las variables anteriores, podemos calcular el valor de R:

$$R = \frac{V_{CC} - V_{OL} - V_{LED}}{I_{LED}}$$

$$= (5.0 - 0.37 - 1.6)\ \text{V}/10\ \text{mA} = 303\ \Omega$$

Cabe mencionar que usted no tiene que utilizar una salida de drenaje abierto para controlar un LED. La figura 3-53(a) muestra un LED que está controlado por una salida de compuerta NAND CMOS con arranque activo. Si ambas entradas son ALTO, los transistores inferiores (de canal-n) llevarán la salida al nivel BAJO como en la versión de drenaje abierto. Si cualquier entrada es BAJO, la salida será ALTO, aunque uno o ambos transistores superiores (de canal-p) estén encendidos, no circulará corriente a través del LED.

En algunas familias CMOS, se puede "encender" un LED cuando la salida se encuentra en el estado ALTO, como se muestra en la figura 3-53(b). Esto es posible si la salida puede suministrar una cantidad considerable de corriente para satisfacer los requerimientos del LED. Sin embargo, el método (b) no se emplea tan frecuentemente como el (a), porque la mayoría de las salidas CMOS y TTL no pueden suministrar tanta corriente en el estado ALTO como la que pueden consumir en el estado BAJO.

VALORES DE RESISTOR

En la mayoría de las aplicaciones, el valor preciso de los resistores en serie de LED no es importante, mientras que grupos de LED cercanos tengan controladores semejantes y resistores para dar un brillo aparentemente igual. En el ejemplo de esta subsección, se puede utilizar un valor de resistor estándar de 270, 300 o 330 ohms, cualquiera que sea fácil de conseguir.

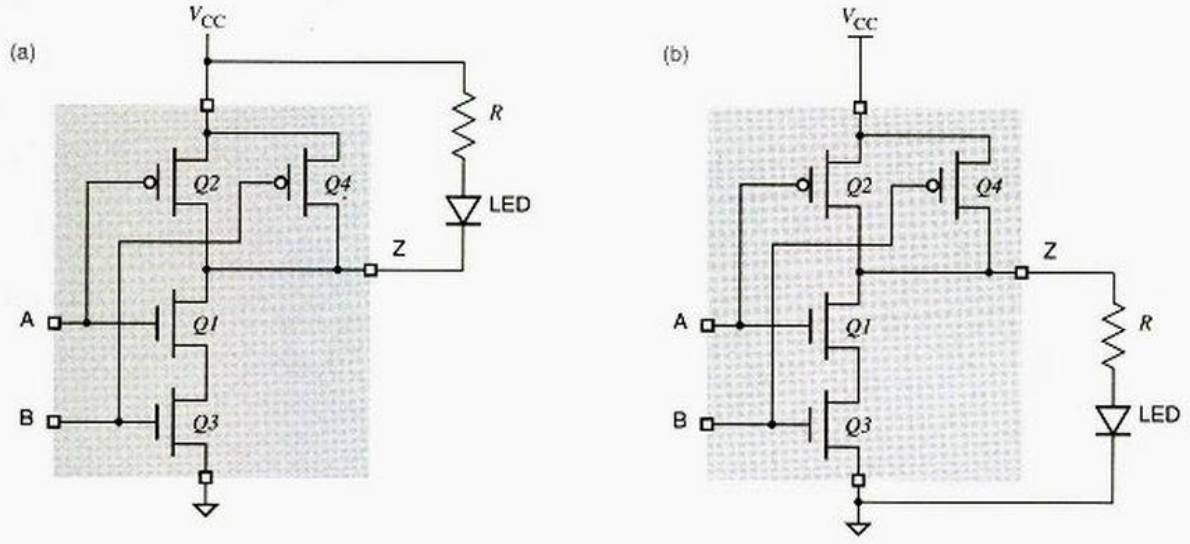


Figura 3-53 Control de un LED con una salida CMOS ordinaria: a) corriente consumida, "encendido" en el estado BAJO; b) corriente suministrada, "encendido" en el estado ALTO.

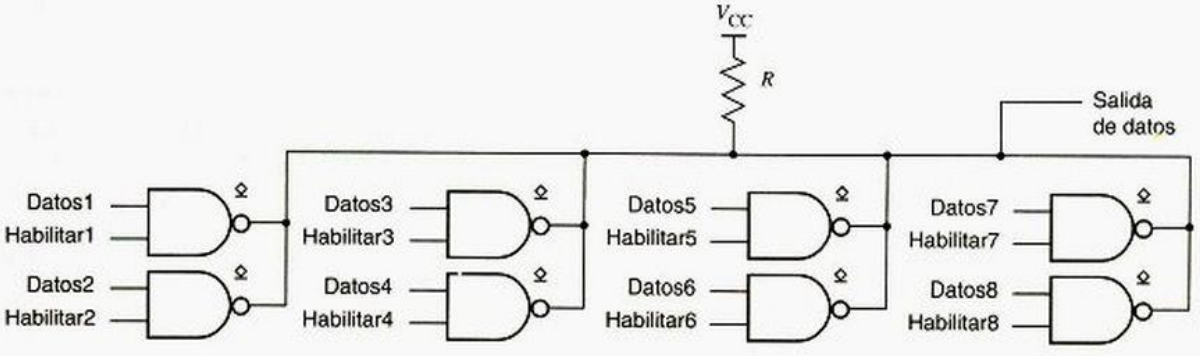
*3.7.6 Buses de fuente múltiple

bus de drenaje abierto

Las salidas de drenaje abierto pueden unirse entre sí para permitir que varios dispositivos, uno a la vez, envíe información a un bus común. En cualquier momento todas las salidas en el bus (excepto una) se encuentran en estado ALTO (abierto). La salida restante ya sea que permanezca en el estado ALTO o que lleve al bus a BAJO, dependiendo de si quiere transmitir un 1 lógico o un 0 lógico en el bus. Los circuitos de control seleccionan el dispositivo particular que puede controlar el bus en cualquier momento.

Por ejemplo, en la figura 3-54, ocho salidas de una compuerta NAND con drenaje abierto de 2 entradas controlan un bus común. La entrada superior de cada compuerta NAND es un bit de datos, y la entrada inferior de cada compuerta es un bit de control.

Figura 3-54 Ocho salidas de drenaje abierto que controlan un bus.



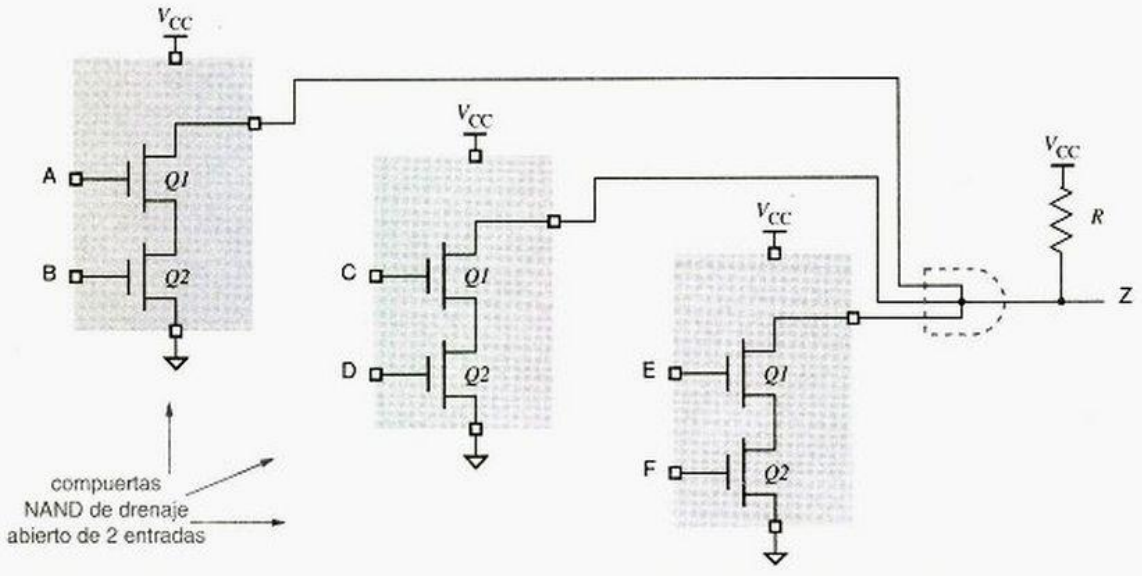


Figura 3-55 Función AND alambrada sobre tres salidas de compuerta NAND de drenaje abierto.

Como máximo un bit de control es ALTO en cualquier momento, lo que habilita al correspondiente bit de datos para que pase a través del bus. (En realidad, el complemento del bit de datos es el que se coloca en el bus). Las demás salidas de la compuerta están en nivel ALTO, es decir, "abiertas", de manera que la entrada de datos de la compuerta habilitada determina el valor del bus.

*3.7.7 Lógica alambrada

lógica alambrada

Si las salidas de varias compuertas con drenaje abierto se conectan entre sí con un resistor de arranque, entonces se realiza la *lógica de alambrado*. Se obtiene una función AND puesto que la salida alambrada está en ALTO si y sólo si todas las salidas de la compuerta individual están en ALTO (en realidad, están abiertas); cualquier salida en nivel BAJO es suficiente para llevar a la salida alambrada al nivel BAJO. La figura 3-55 muestra una función *alambrada AND* con tres entradas. Si cualquiera de las compuertas individuales NAND de 2 entradas tiene ambas entradas en ALTO, esto lleva a la salida alambrada al nivel BAJO; de otro modo, el resistor de arranque R lleva a la salida alambrada a ALTO.

AND alambrado

Cabe indicar que la lógica de alambrado no puede realizarse si se utilizan compuertas que tienen arranque activo (active pull-up). Cuando se conectan dos salidas de este tipo y ambas intentan mantener valores lógicos opuestos, se genera un flujo de corriente muy elevado y un voltaje de salida anormal. La figura 3-56 muestra esta situación, que en ocasiones se denomina conflicto. El voltaje exacto de salida dependerá de las "fuerzas" relativas de los transistores en conflicto, pero con los dispositivos CMOS de 5 V es por lo regular de aproximadamente 1-2 V, casi siempre un voltaje no lógico. Peor aún, si las salidas pelean continuamente durante varios segundos, ¡los chips pueden calentarse lo suficiente para provocar un daño interno y quemar sus dedos!

conflicto

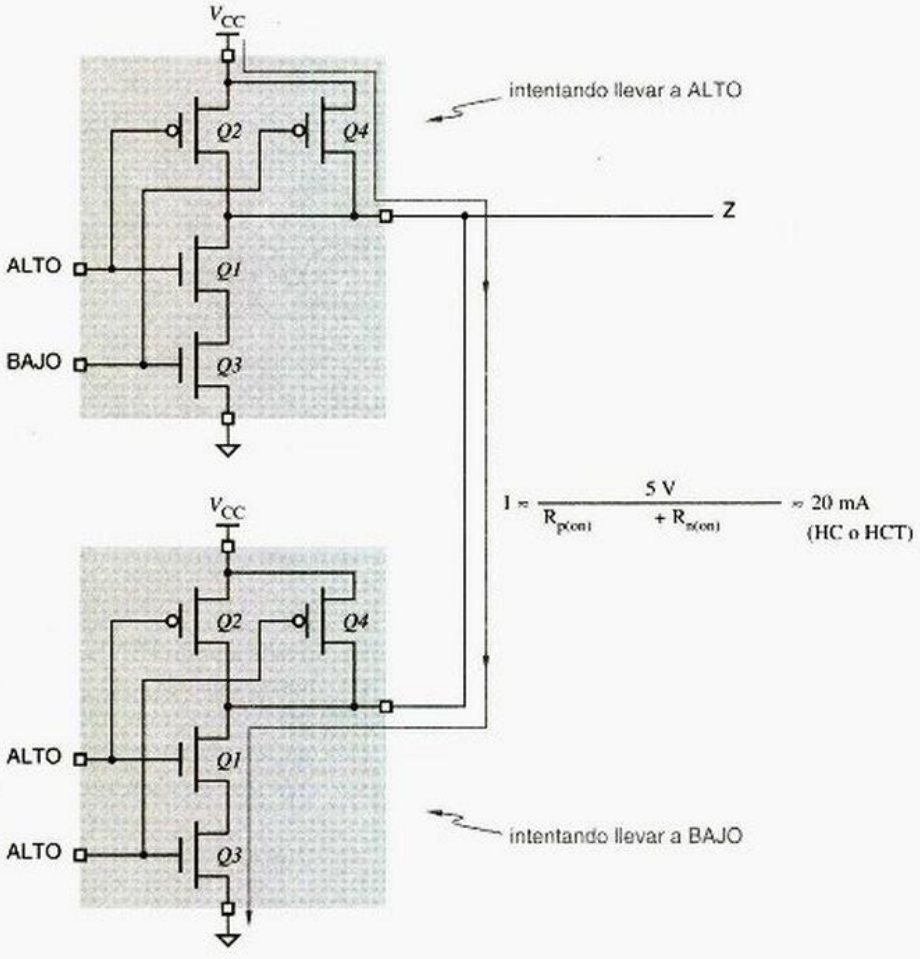


Figura 3-56
Dos salidas CMOS que intentan mantener valores lógicos opuestos en la misma línea.

*3.7.8 Resistores de arranque

cálculo del resistor de arranque

Debe hacerse una adecuada elección del valor para el resistor de arranque R en aplicaciones con drenaje abierto. Se efectúan dos cálculos para clasificar los valores permisibles de R:

Mínimo La suma de la corriente a través de R en el estado **BAJO** y las corrientes de entrada del estado **BAJO** de las compuertas que controlan las salidas alambradas no debe exceder la capacidad de control de estado **BAJO** de la salida activa, 4 mA para la serie CMOS HC y HCT, 24 mA para CA y ACT.

Máximo La caída de voltaje a través de R en el estado **ALTO** no debe reducir el voltaje de salida por debajo de 2.4 V, que es V_{IHmin} para las compuertas controladas típicas, más un margen de ruido de 400 mV. Esta caída es producida por la corriente de fuga de salida en estado **ALTO** de las salidas alambradas y las corrientes de entrada de estado **ALTO** de las compuertas controladas.

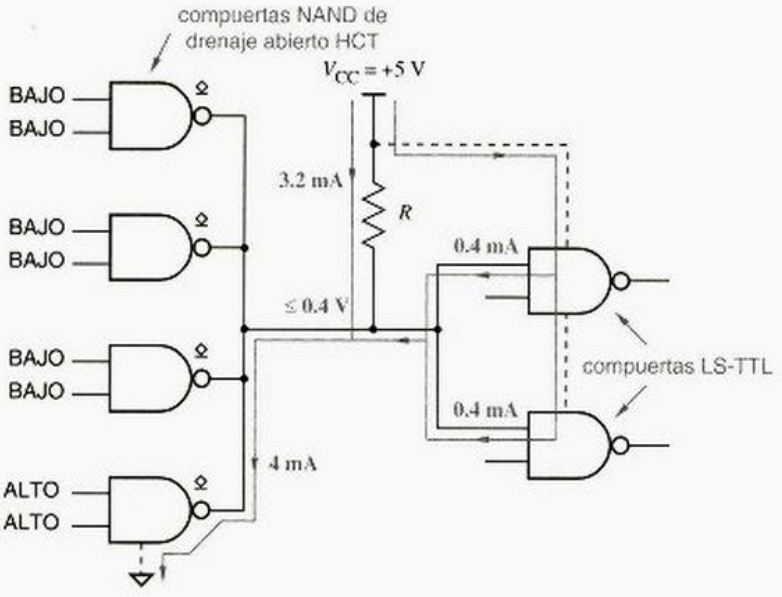


Figura 3-57
Cuatro salidas de drenaje abierto que controlan dos entradas en el estado BAJO.

Por ejemplo, supongamos que cuatro salidas de drenaje abierto HCT se conectan entre sí y manejan dos entradas LS-TTL (sección 3.11) como se muestra en la figura 3-57. Una salida en BAJO debe consumir 0.4 mA de cada entrada LS-TTL, también debe consumir la corriente a través del resistor de arranque R. Para que la corriente total permanezca dentro de la especificación de 4 mA HCT $I_{OL\max}$, la corriente que circula a través de R no puede ser mayor que

$$I_{R(\max)} = 4 - (2 \cdot 0.4) = 3.2 \text{ mA}$$

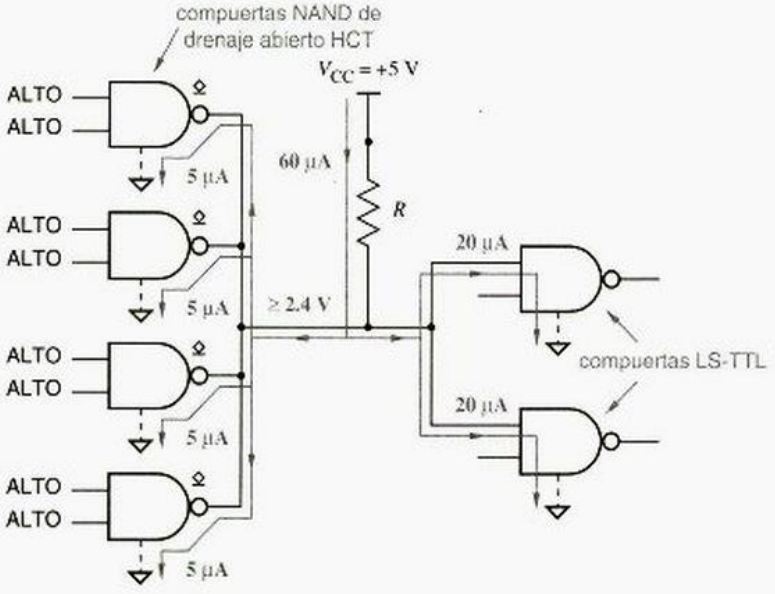


Figura 3-58
Cuatro salidas de drenaje abierto que controlan dos entradas en el estado ALTO.

Suponiendo que V_{OL} de la salida de drenaje abierto es 0.0 V, el valor mínimo de R es

$$R_{min} = (5.0 - 0.0)/I_{R(max)} = 1562.5\ \Omega$$

En el estado ALTO, las salidas típicas de drenaje abierto tienen una corriente de fuga máxima de 5 μA, y las entradas típicas LS-TTL requieren 20 μA de corriente de fuente. Por lo tanto, el requerimiento de corriente de estado ALTO, como se muestra en la figura 3-58 es:

$$I_{R(fuga)} = (4 \cdot 5) + (2 \cdot 20) = 60\ \mu A$$

Esta corriente produce una caída de voltaje a través de R, y no debe ser inferior al voltaje de salida ($V_{OHmin} = 2.4$ V); de este modo el valor máximo de R es

$$R_{max} = (5.0 - 2.4)/I_{R(fuga)} = 43.3\ k\Omega$$

Por lo tanto, puede utilizarse cualquier valor de R que esté comprendido entre 1562.5 Ω y 43.3 kΩ. Valores mayores provocan una reducción en el consumo de energía y mejoran el margen de ruido de estado BAJO, mientras que valores inferiores incrementan el consumo de energía pero mejoran tanto el margen de ruido del estado ALTO como la velocidad de las transiciones de salida BAJO a ALTO.

SUPOSICIÓN DE DRENAJE ABIERTO

En nuestros cálculos del resistor de drenaje abierto, supusimos que el voltaje de salida puede ser tan bajo como 0.0 V en lugar de 0.4 V ($V_{OL,max}$) a fin de obtener un resultado en el peor de los casos. Es decir, incluso si la salida de drenaje abierto es tan fuerte que pueda llevar al voltaje de salida hasta 0.0 V (solamente se requiere para disminuir hasta 0.4 V), nunca le permitimos consumir más de 4 mA, de modo que no llegue a forzarse demasiado. Algunos diseñadores prefieren utilizar 0.4 V en este cálculo, suponiendo que si la salida es tan buena que pueda llevarla más debajo de 0.4 V, una pequeña cantidad de corriente de consumo en exceso más allá de 4 mA no la dañará.

3.8 Familias lógicas CMOS

CMOS serie 4000

La primer familia CMOS que tuvo éxito comercial fue la *serie 4000 de CMOS*. Aun cuando los circuitos de la serie 4000 ofrecían los beneficios de bajo consumo de energía, eran bastante lentos y no eran fáciles de interconectar con la familia lógica más popular de esa época, la familia bipolar TTL. Así, la serie 4000 fue sustituida en la mayoría de las aplicaciones por las familias más capaces de CMOS que se analizan en esta sección.

Los números de parte de todos los dispositivos CMOS que analizaremos tienen la forma "74FAM*nn*", donde "FAM" es un mnemónico alfabético de la familia y *nn* es un designador numérico de función. Los dispositivos de las diferentes familias que tienen el mismo valor de *nn* realizan la misma función. Por ejemplo, la 74HC30, 74HCT30, 74AC30, 74ACT30 y 74AHC30 son todos compuertas NAND de 8 entradas.

El prefijo "74" es simplemente un número que fue empleado por un antiguo y popular proveedor de dispositivos TTL, Texas Instruments. El prefijo "54" se usa para partes idénticas cuya operación esta especificada en un intervalo más amplio de temperatura y voltajes de alimentación, para su uso en aplicaciones militares. En general estas partes se fabrican en la misma forma que sus contrapartes de la serie 74, la única excepción es que

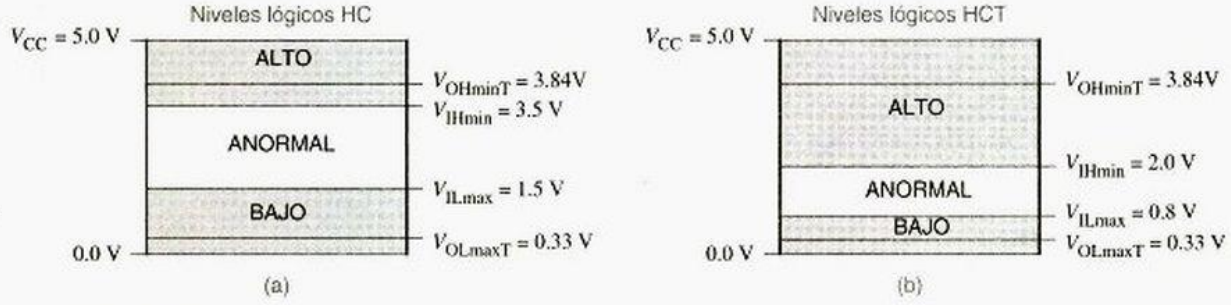


Figura 3-59 Niveles de entrada y salida para dispositivos CMOS utilizando una alimentación de 5 V: a)HC; b) HCT.

se prueban, examinan e identifican en forma diferente, se genera mucho más trabajo de escritorio y se cobra un precio más alto, naturalmente.

3.8.1 HC y HCT

HC (CMOS de alta velocidad)

HCT (CMOS de alta velocidad, compatible con TTL)

Las primeras dos familias CMOS de la serie 74 son la *HC (High-speed CMOS, CMOS de alta velocidad)* y *HCT (High-speed CMOS, TTL compatible; CMOS de alta velocidad, compatible con TTL)*. En comparación con la familia 4000 original, ambas, HC y HCT, tienen mayor velocidad y mejor capacidad de consumo y suministro de corriente. La familia HCT utiliza un voltaje de alimentación V_{CC} de 5 V y puede conectarse a dispositivos TTL, que también utilizan un voltaje de alimentación de 5 V.

La familia HC está optimizada para utilizarse en los sistemas que manejan exclusivamente lógica CMOS, y pueden emplear cualquier voltaje de alimentación entre 2 y 6 V. Se utiliza un voltaje mayor para obtener mayor velocidad y un voltaje menor para menor disipación de energía. La disminución del voltaje de alimentación es especialmente efectiva, puesto que la mayor parte del consumo de energía (en un dispositivo CMOS) es proporcional al cuadrado del voltaje (si consideramos que la potencia es igual al producto CV^2f).

Incluso cuando se utiliza una fuente de alimentación de 5 V, los dispositivos HC no son completamente compatibles con los dispositivos TTL. En especial, los circuitos HC están diseñados para reconocer niveles de entrada CMOS. Suponiendo que tengamos un voltaje de alimentación de 5.0 V, la figura 3-59(a) muestra los niveles de entrada y de salida de los dispositivos HC. Los niveles de salida que producen los dispositivos TTL no coinciden exactamente con este intervalo, de modo que los dispositivos HCT utilizan los diferentes niveles de entrada que se muestran en (b). Estos niveles se establecen durante el proceso de fabricación cuando se fabrican transistores con diferentes umbrales de conmutación, lo que produce las diferentes características de transferencia que se indican en la figura 3-60.

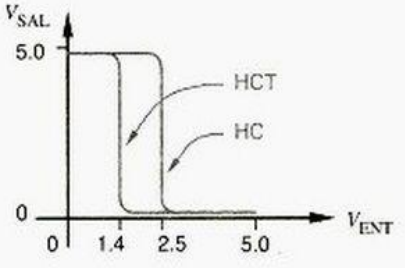


Figura 3-60
Características de transferencia de circuitos HC y HCT bajo condiciones típicas.

Tendremos más qué decir acerca de la interfaz CMOS/TTL en la sección 3.12. Por ahora, nos interesa advertir que los dispositivos HC y HCT son esencialmente idénticos en sus especificaciones de salida; sólo difieren sus niveles de entrada.

3.8.2 VHC y VHCT

En las décadas de 1980 y 1990 se lanzaron al mercado varias familias nuevas de circuitos CMOS. Dos de las más recientes y probablemente las más versátiles son las *VHC (Very High-Speed CMOS:* CMOS de muy alta velocidad) y VHCT *(Very High-Speed CMOS, TTL compatible: CMOS de muy alta velocidad compatible con TTL).* Estas familias son aproximadamente el doble de rápidas que las HC/HCT, mientras mantienen la compatibilidad con sus predecesores. Como con HC y HCT, las familias VHC y VHCT difieren entre sí solamente en los niveles de entrada que reconocen; sus características de salida son las mismas.

VHC (CMOS de muy alta velocidad)

VHCT (CMOS de muy alta velocidad, compatible con TTL)

También como HC/HCT, las salidas VHC/VHCT tienen *control de salida simétrico.* Es decir, una salida puede consumir o suministrar iguales cantidades de corriente; la salida es así de "fuerte" en ambos estados. Otras familias lógicas, incluyendo las familias FCT y TTL introducidas posteriormente, tienen *control de salida asimétrico*; pueden consumir mucha más corriente en el estado BAJO que la que pueden suministrar en el estado ALTO.

control de salida simétrico

control de salida asimétrico

3.8.3 Características eléctricas de HC, HCT, VHC y VHCT

Las características eléctricas de las familias HC, HCT, VHC y VHCT se resumen en esta subsección. Las especificaciones suponen que los dispositivos se utilizan con una alimentación nominal de 5 V, aunque es posible la operación (reducida) con cualquier voltaje de alimentación en el intervalo 2-5.5 V (hasta 6 V para HC/HCT). Haremos un examen más detallado para el funcionamiento de bajo voltaje y voltaje mezclado en la sección 3.13.

Las partes comerciales (serie 74) están destinadas a funcionar a temperaturas entre los 0°C y los 70°C, mientras que las partes militares (serie 54) se caracterizan por funcionar entre –55°C y 125°C. Las especificaciones en la tabla 3-5 suponen una temperatura de operación de 25°C. Una hoja completa de datos del fabricante proporciona especificaciones adicionales para el funcionamiento del dispositivo sobre todo el intervalo de temperatura.

La mayoría de los dispositivos dentro de una familia lógica dada tienen las mismas especificaciones eléctricas para entradas y salidas, difiriendo por lo regular solamente en el consumo de energía y el retardo de propagación. La tabla 3-5 incluye especificaciones para una compuerta NAND de dos entradas 74x00 y un decodificador de 3 a 8 74x138 en las familias HC, HCT, VHC y VHCT. La compuerta NAND '00 se incluye como el bloque más pequeño de construcción de diseño lógico en cada familia, mientras que el '138 es una parte de "mediana escala" que contiene el equivalente de aproximadamente 15 compuertas NAND. (La especificación del '138 se incluye para permitir la comparación con

VERY = ADVANCED, TIPOS DE

Las familias lógicas VHC y VHCT son fabricadas por diversas compañías, incluyendo Motorola, Fairchild y Toshiba. Familias compatibles con especificaciones semejantes pero no idénticas se fabrican por Texas Instruments y Philips; se conocen como AHC y AHCT, donde la "A" significa "Avanzado" (en inglés, "Advanced").

Tabla 3-5 Características de velocidad y energía de las familias CMOS funcionando a 5 V.

Descripción	Parte	Símbolo	Condición	Familia HC	Familia HCT	Familia VHC	Familia VHCT
Retardo de propagación típico (ns)	'00	t_{PD}		9	10	5.2	5.5
	'138			18	20	7.2	8.1
Corriente de alimentación en reposo (μA)	'00	I_{CC}	$V_{in} = 0$ o V_{CC}	2.5	2.5	5.0	5.0
	'138		$V_{in} = 0$ o V_{CC}	40	40	40	402
Disipación de energía en reposo (mW)	'00		$V_{in} = 0$ o V_{CC}	0.0125	0.0125	0.025	0.025
	'138		$V_{in} = 0$ o V_{CC}	0.2	0.2	0.2	0.2
Capacitancia de disipación de energía (pF)	'00	C_{PD}		22	15	19	17
	'138	C_{PD}		55	51	34	49
Disipación dinámica de energía (mW/MHz)	'00			0.55	0.38	0.48	0.43
	'138			1.38	1.28	0.85	1.23
Disipación de energía total (mW)	'00		$f = 100$ kHz	0.068	0.050	0.073	0.068
	'00		$f = 1$ MHz	0.56	0.39	0.50	0.45
	'00		$f = 10$ MHz	5.5	3.8	4.8	4.3
	'138		$f = 100$ kHz	0.338	0.328	0.285	0.323
	'138		$f = 1$ MHz	1.58	1.48	1.05	1.43
	'138		$f = 10$ MHz	14.0	13.0	8.7	12.5
Producto velocidad-energía (pJ)	'00		$f = 100$ kHz	0.61	0.50	0.38	0.37
	'00		$f = 1$ MHz	5.1	3.9	2.6	2.5
	'00		$f = 10$ MHz	50	38	25	24
	'138		$f = 100$ kHz	6.08	6.55	2.05	2.61
	'138		$f = 1$ MHz	28.4	29.5	7.56	11.5
	'138		$f = 10$ MHz	251	259	63	101

la familia más rápida FCT en la sección 3.8.4; las compuertas '00 no están fabricadas en la familia FCT.)

El primer renglón de la tabla 3-5 especifica el retardo de propagación. Como se discutió en la sección 3.6.2, dos números, t_{pHL} y t_{pLH}, pueden utilizarse para especificar el retardo; el número en la tabla es el peor de los dos casos. Saltando a la tabla 3-11 en la página 167, se puede ver que HC y HCT tienen aproximadamente la misma velocidad que LS TTL, y que VHC y VHCT son casi tan rápidos como ALS TTL. El retardo de

NOTA ACERCA DE NOTACIÓN La "x" en la notación "74x00" toma el lugar de un denominador de familia tal como HC, HCT, VHC, VHCT, FCT, LS, ALS, AS o F. También podemos referirnos a una parte genérica de esta clase simplemente como un " '00" y dejar fuera el "74x".

OBTENGA DISCRETAMENTE MÁS

Los circuitos VHC y VHCT también pueden ser controlados por dispositivos TTL, los cuales pueden producir niveles de salida ALTO tan bajos como de 2.4 V. Como explicamos en la sección 3.5.3, una salida CMOS puede extraer corriente adicional desde la fuente de alimentación si cualquiera de las entradas son no ideales. En el caso de un inversor HCT o VHCT con una entrada en ALTO de 2.4 V, el transistor inferior de salida de canal-*n* está completamente "encendido". Sin embargo, el transistor superior, de canal-*p*, se encuentra también parcialmente "encendido". Esto permite el flujo adicional de corriente en reposo, especificada como ΔI_{CC} o I_{CCT} en la hoja de datos, la que puede ser tanto como 2-3 mA por entrada no ideal en dispositivos HCT y VHCT.

propagación para el '138 es un tanto más largo que para el '00, puesto que las señales deben viajar a través de tres o cuatro niveles de compuertas internamente.

El segundo y tercer renglones de la tabla muestran que la disipación de energía en reposo de estos dispositivos CMOS es prácticamente nula, por debajo de un miliwatt (mW) si las entradas tienen niveles CMOS (0 V para BAJO y V_{CC} para ALTO). (Nótese que en la tabla, los números de la disipación de energía en reposo dados para el '00 son por compuerta, mientras que para el '138 se aplican al dispositivo MSI por completo).

Como discutimos en la sección 3.6.3 la disipación dinámica de energía en una compuerta CMOS depende de la excursión del voltaje de la salida (por lo regular, V_{CC}), la frecuencia de transición de salida (f) y la capacitancia que está siendo cargada y descargada en las transiciones, de acuerdo con la fórmula

$$P_D = (C_L + C_{PD}) \cdot V_{DD}^2 \cdot f$$

Aquí, C_{PD} es la capacitancia de consumo de energía del dispositivo y C_L es la capacitancia de la carga unida a la salida CMOS en una aplicación dada. La tabla enumera tanto C_{PD} como un factor de disipación dinámica de energía equivalente en unidades de miliwatts por megahertz, suponiendo que $C_L = 0$. Haciendo uso de este factor, la disipación de energía total se calcula a varias frecuencias como la suma de la disipación dinámica de energía a esa frecuencia y la disipación de energía en reposo.

producto velocidad-energía

Acto seguido en la tabla, se muestra el *producto de velocidad-energía* que es simplemente el producto del retardo de propagación y el consumo de energía de una compuerta típica; el resultado se mide en picojoules (pJ). Recuerde que, como estudió en sus clases de física, el joule es una unidad de energía, de modo que el producto velocidad-energía mide una clase de eficiencia: cuánta energía utiliza una compuerta lógica para conmutar su salida. En estos tiempos, es obvio que entre menor sea el gasto de energía, será mejor.

AHORRO DE ENERGÍA

Existen razones tanto prácticas como geopolíticas para ahorrar energía en los sistemas digitales. El consumo de energía más bajo significa menor costo de fuentes de alimentación y sistemas de enfriamiento. También, se mejora más la confiabilidad de un sistema digital al funcionar a menor temperatura que por cualquier otra estrategia simple de optimación de confiabilidad.

Tabla 3-6 Especificaciones de entrada para familias CMOS con V_{CC} entre 4.5 y 5.5 V.

Descripción	Símbolo	Condición	Familia			
			HC	HCT	VHC	VHCT
Corriente de fuga de entrada (μA)	I_{Imax}	V_{ent} = cualquier	±1	±1	±1	±1
Capacitancia de entrada máxima (pF)	C_{INmax}		10	10	10	10
Voltaje de entrada de nivel BAJO (V)	V_{ILmax}		1.35	0.8	1.35	0.8
Voltaje de entrada de nivel ALTO (V)	V_{IHmin}		3.85	2.0	3.85	2.0

La tabla 3-6 proporciona las especificaciones de entrada de dispositivos CMOS típicos en cada una de las familias. Algunas de las especificaciones suponen que la alimentación de 5 V tiene un margen de ± 10%; es decir, V_{CC} puede estar en cualquier lugar entre 4.5 y 5.5 V. Estos parámetros fueron discutidos en secciones anteriores, pero como referencia se resumen aquí sus significados:

I_{Imax} La corriente de entrada máxima para cualquier valor de voltaje de entrada. Esta especificación establece que la corriente que fluye hacia dentro o hacia fuera de una entrada CMOS es 1 μA o menor para cualquier valor del voltaje de entrada. En otras palabras, las entradas CMOS casi no crean carga de CD en los circuitos que controlan.

C_{INmax} La capacitancia máxima de una entrada. Este número puede ser utilizado cuando se supone la carga de CA en una salida que controla ésta y otras entradas. La mayoría de los fabricantes también especifican una capacitancia de entrada típica, más baja de alrededor de 5 pF, la cual proporciona una buena estimación de la carga de CA si usted no tiene mala suerte.

V_{ILmax} El voltaje máximo que se garantiza para que una entrada lo reconozca como BAJO. Advierta que los valores son diferentes para HC/VHC en comparación con HCT/VHCT. El valor de "CMOS", 1.35 V, es 30% del mínimo voltaje de alimentación, mientras que el valor "TTL" es 0.8 V por compatibilidad con las familias TTL.

DISIPACIÓN DE ENERGÍA CMOS Y TTL

A frecuencias de transición altas (f), las familias CMOS en realidad utilizan más energía que TTL. Por ejemplo, compare CMOS HCT en la tabla 3-5 para f = 10 MHz con LS TTL en la tabla 3-11; una compuerta CMOS utiliza tres veces tanta energía como una compuerta TTL a esta frecuencia. Tanto HCT como LS pueden ser empleadas en sistemas con frecuencias de "reloj" máximas de hasta aproximadamente 20 MHz, de modo que se puede pensar que CMOS no es tan bueno para sistemas de alta velocidad. Sin embargo, las frecuencias de transición de la mayoría de las salidas en sistemas típicos son mucho menores que la frecuencia máxima presente en el sistema (por ejemplo, véase el ejercicio 3.77). De este modo, los sistemas CMOS típicos tienen una disipación de energía total menor que la que tendrían si estuvieran construidos con TTL.

Tabla 3-7 Especificaciones de salida para familias CMOS funcionando con V_{CC} entre 4.5 y 5.5 V.

Descripción	Símbolo	Condición	Familia			
			HC	HCT	VHC	VHCT
Corriente de salida de nivel BAJO (mA)	I_{OLmaxC}	Carga CMOS	0.02	0.02	0.05	0.05
	I_{OLmaxT}	Carga TTL	4.0	4.0	8.0	8.0
Voltaje de salida de nivel BAJO (V)	V_{OLmaxC}	$I_{sal} \leq I_{OLmaxC}$	0.1	0.1	0.1	0.1
	V_{OLmaxT}	$I_{sal} \leq I_{OLmaxT}$	0.33	0.33	0.44	0.44
Corriente de salida de nivel ALTO (mA)	I_{OHmaxC}	Carga CMOS	−0.02	−0.02	−0.05	−0.05
	I_{OHmaxT}	Carga TTL	−4.0	−4.0	−8.0	−8.0
Voltaje de salida de nivel ALTO (V)	V_{OHminC}	$\|I_{ent}\| \leq \|I_{OHmaxC}\|$	4.4	4.4	4.4	4.4
	V_{OHminT}	$\|I_{ent}\| \leq \|I_{OHmaxT}\|$	3.84	3.84	3.80	3.80

V_{IHmin} El voltaje mínimo que se garantiza que una entrada reconocerá como ALTO. El "valor" CMOS, 3.85 V, es un 70% del voltaje de alimentación máximo, mientras que el valor "TTL" es de 2.0 V para compatibilidad con las familias TTL. (A diferencia de los niveles CMOS, los niveles de entrada TTL no son simétricos con respecto a los rieles de alimentación.)

Las especificaciones para las salidas CMOS compatibles con TTL por lo regular tienen dos conjuntos de parámetros de salida; un conjunto o el otro se utilizan dependiendo de cómo está cargada una salida. Una *carga CMOS* es una que requiere que la salida consuma y suministre muy poca corriente de CD, 20 μA para HC/HCT y 50 μA para VHC/VHCT. Éste es, por supuesto, el caso cuando las salidas CMOS controlan solamente entradas CMOS. Con cargas CMOS, las salidas CMOS mantienen un voltaje de salida dentro de 0.1 V de las líneas de alimentación, 0 y V_{CC}. (Un caso límite, V_{CC} = 4.5 V se utiliza para las entradas de la tabla; de aquí, V_{OHminC} = 4.4 V.)

carga CMOS

Una *carga TTL* consume mucha más corriente de consumo y de fuente (consumo y suministro), hasta 4 mA desde una salida HC/HCT y 8 mA desde una salida VHC/VHCT. En este caso, se presenta una mayor caída de voltaje a través de los transistores "encendidos" en el circuito de salida, pero el voltaje de salida todavía se garantiza que esté dentro del intervalo normal de los niveles de salida TTL.

carga TTL

La tabla 3-7 enumera las especificaciones de salida CMOS tanto para cargas CMOS como TTL. Estos parámetros tienen los significados siguientes:

I_{OLmaxC} La corriente máxima que puede suministrar una salida en el estado BAJO mientras controla una carga CMOS. Puesto que éste es un valor positivo, la corriente fluye *hacia dentro* de la terminal de salida.

I_{OLmaxT} La corriente máxima que puede suministrar una salida en el estado BAJO mientras controla una carga TTL.

V_{OLmaxC} El máximo voltaje que se garantiza que producirá una salida al nivel BAJO mientras controla una carga CMOS, es decir, mientras que no se exceda I_{OLmaxC}.

V_{OLmaxT} El máximo voltaje que se garantiza que producirá una salida al nivel BAJO mientras controla una carga TTL, es decir, mientras que no se exceda I_{OLmaxT}.

I_{OHmaxC} La corriente máxima que puede suministrar una salida en el estado ALTO mientras controla una carga CMOS. Puesto que éste es un valor negativo, la corriente positiva fluye hacia afuera de la terminal de salida.

I_{OHmaxT} La corriente máxima que puede suministrar una salida en el estado ALTO mientras controla una carga TTL.

V_{OHminC} El voltaje mínimo que se garantiza producirá una salida en ALTO mientras controla una carga CMOS, es decir, mientras no se exceda I_{OHmaxC}.

V_{OHminT} El voltaje mínimo que se garantiza producirá una salida en ALTO mientras controla una carga TTL, es decir, mientras no se exceda I_{OHmaxC}.

Los parámetros de voltaje anteriores determinan los márgenes de ruido de CD. El margen de ruido de CD del estado BAJO es la diferencia entre V_{OLmax} y V_{ILmax}. Esto depende de las características tanto de la salida que controla como de las entradas controladas. Por ejemplo, el margen de ruido de CD del estado BAJO del HCT controlando unas cuantas entradas HCT (una carga CMOS) es $0.8 - 0.1 = 0.7$ V. Con una carga TTL, el margen de ruido para las entradas HCT decae a $0.8 - 0.33 = 0.47$ V. De manera semejante, el margen de ruido de CD del estado ALTO es la diferencia entre V_{OHmin} y V_{IHmin}. En general, cuando se interconectan familias diferentes, se tiene que comparar los V_{OLmax} y V_{OHmin} apropiados de la compuerta que controla con V_{ILmax} y V_{IHmin} de todas las compuertas controladas para determinar los márgenes de ruido del peor de los casos.

Los parámetros I_{OLmax} e I_{OHmax} en la tabla determinan la capacidad de fan out de salida y son especialmente importantes cuando una salida controla entradas en una o más diferentes familias. Deben realizarse dos cálculos para determinar si una salida se encuentra funcionando dentro de su capacidad de fan out nominal:

fan out en estado ALTO	Se suman los valores I_{IHmax} para todas las entradas controladas. La suma debe ser menor que I_{OHmax} de la salida que controla.
fan out en estado BAJO	Se suman los valores I_{ILmax} para todas las entradas controladas. La suma debe ser menor que I_{OLmax} de la salida que cotrola.

Nótese que las características de entrada y salida de componentes específicos pueden variar a partir de los valores representativos dados en la tabla 3-7, de modo que usted siempre consulte la hojas de datos del fabricante cuando se analice un diseño real.

*3.8.4 FCT y FCT-T

FCT (CMOS rápido, compatible con TTL)

A principios de la década de 1990 fue presentada otra familia CMOS. El beneficio clave de la familia *FCT (Fast CMOS, TTL compatible: CMOS rápido, compatible con TTL)* era su capacidad para satisfacer o exceder la velocidad y la capacidad de control de salida de las mejores familias TTL, mientras que reducía el consumo de energía y mantenía una total compatibilidad con TTL.

La familia original FCT tenía la desventaja de producir un V_{OH} CMOS de 5 V, creando una enorme disipación CV^2f de energía y ruido de circuito a medida que sus salidas oscilaban desde 0 V hasta casi 5 V en aplicaciones de alta velocidad (25 MHz+). Una variación de la familia, *FCT-T (Fast CMOS, TTL compatible with TTL V_{OH})* fue rápidamente introducida con innovaciones de circuito para reducir el voltaje de salida de nivel ALTO, reduciendo por ello tanto el consumo de energía y el ruido de conmutación mientras que mantenía la misma alta velocidad de operación que el FCT original. Un sufijo de "T" se emplea en los números de parte para denotar la estructura de salida FCT-T, por ejemplo, 74FCT138T contra 74FCT138.

FCT-T (CMOS rápido, compatible con TTL con TTL V_{OH})

La familia FCT-T sigue siendo muy popular en la actualidad. Una aplicación clave de FCT-T se encuentra controlando buses y otras cargas pesadas. En comparación con otras familias CMOS, puede suministrar o consumir picos de corriente, de hasta 64 mA en estado BAJO.

*3.8.5 Características eléctricas del FCT-T

Las características eléctricas de la familia FCT-T de 5 V se resumen en la tabla 3-8. La familia se encuentra específicamente diseñada para entremezclarse con dispositivos TTL, de modo que su operación está solamente especificada con una alimentación nominal de 5 V y niveles lógicos TTL. Algunos fabricantes están comenzando a vender partes con capacidades semejantes empleando una alimentación de 3.3 V, y utilizando la designación FCT. Sin embargo, son diferentes dispositivos con números de partes diferentes.

Las compuertas lógicas individuales no son fabricadas en la familia FCT. Quizás el elemento lógico FCT más simple es un decodificador 74FCT138T, el cual tiene seis entradas, ocho salidas y contiene el equivalente de aproximadamente una docena de compuertas de 4 entradas internamente. (Esta función se describe posteriormente, en la sección 5.4.4.) Comparando este retardo de propagación y consumo de energía en la tabla 3-8 con los números correspondientes HCT y VHCT de la tabla 3-5 en la página 138, se aprecia que la familia FCT-T es superior tanto en velocidad como en disipación de energía. Cuando se comparan, advierta que los fabricantes de FCT-T especifican únicamente retardos de propagación máximos, no típicos.

A diferencia de otras familias CMOS, la FCT-T no tiene una especificación C_{PD}. En su lugar, tiene una especificación I_{CCD}:

I_{CCD} es la corriente de alimentación dinámica, en unidades de mA/MHz. Ésta es la cantidad de corriente de alimentación adicional que fluye cuando una entrada está cambiando a la velocidad de 1 MHz.

CONMUTACIÓN EXTREMA

Las salidas de dispositivos en las familias FCT y FCT-T tienen muy baja impedancia y, en consecuencia, tiempos de ascenso y caída extremadamente rápidos. De hecho, son tan rápidos que son con frecuencia una fuente principal de problemas "analógicos", incluyendo ruido de conmutación y "rebote de tierra", de manera que debe tenerse más cuidado en el diseño analógico y físico de las tarjetas de circuito impreso utilizando éstas y otras partes de muy alta velocidad. Para reducir los efectos de reflejos de línea de transmisión (sección 11.4), otra preocupación del diseño de alta velocidad, algunas salidas FCT-T tienen resistores en serie de 25 Ω integrados.

Tabla 3-8 Especificaciones para un decodificador 74FCT138T en la familia lógica FCT-T.

Descripción	*Símbolo*	*Condición*	*Valor*
Retardo de propagación máximo (ns)	t_{PD}		5.8
Corriente de alimentación en reposo (μA)	I_{CC}	$V_{ent} = 0$ o V_{CC}	200
Disipación de energía en reposo (mW)		$V_{ent} = 0$ o V_{CC}	1.0
Corriente de alimentación dinámica (mA/MHz)	I_{CCD}	Salidas abiertas, una entrada que cambia	0.12
Corriente de alimentación en reposo por entrada TTL (mA)	ΔI_{CC}	$V_{ent} = 3.4$ V	2.0
Disipación de energía total (mW)		$f = 100$ kHz $f = 1$ MHz $f = 10$ MHz	0.60 1.06 1.6
Producto velocidad-energía (pJ)		$f = 100$ kHz $f = 1$ MHz $f = 10$ MHz	6.15 9.3 41
Corriente de fuga de entrada (μA)	I_{Imax}	V_{ent} = cualquiera	±5
Capacitancia de entrada típica (pF)	C_{INtyp}		5
Voltaje de entrada de nivel BAJO (V)	V_{ILmax}		0.8
Voltaje de entrada de nivel ALTO (V)	V_{IHmin}		2.0
Corriente de salida de nivel BAJO (mA)	I_{OLmax}		64
Voltaje de salida de nivel BAJO (V)	V_{OLmax}	$I_{sal} \leq I_{OLmax}$	0.55
Corriente de salida de nivel ALTO (mA)	I_{OHmax}		−15
Voltaje de salida de nivel ALTO (V)	V_{OHmin} V_{OHtyp}	$\lvert I_{sal}\rvert \leq \lvert I_{OHmax}\rvert$ $\lvert I_{sal}\rvert \leq \lvert I_{OHmax}\rvert$	2.4 3.3

La especificación I_{CCD} proporciona la misma información que C_{PD}, pero de una manera diferente. La disipación de energía interna del circuito debida a transiciones a una frecuencia f dada puede calcularse mediante la fórmula

$$P_T = V_{CC} \cdot I_{CCD} \cdot f$$

De este modo, I_{CCD}/V_{CC} es algebraicamente equivalente a la especificación C_{PD} de otras familias CMOS (véase el ejercicio 3.83). FCT-T también tiene una especificación ΔI_{CC} para la corriente en reposo extra que se consume con entradas ALTO no ideales (véase el cuadro en la parte superior de la página 139).

3.9 Lógica bipolar

Las familias lógicas bipolares utilizan diodos semiconductores y transistores de unión bipolar como los bloques de construcción básicos de los circuitos lógicos. Los elementos lógicos bipolares más simples hacen uso de diodos y resistores para realizar operaciones lógicas; esto es llamado lógica de diodos. La mayoría de las compuertas lógicas TTL utilizan internamente lógica de diodos y aceleran su capacidad de control de salida empleando circuitos de transistores. Algunas compuertas TTL utilizan configuraciones en paralelo de transistores para efectuar funciones lógicas. Las compuertas ECL, que se describen en la sección 3.14, emplean transistores como interruptores de corriente para conseguir muy alta velocidad.

lógica de diodo

En esta sección se cubre el funcionamiento básico de los circuitos lógicos bipolares hechos con diodos y transistores, mientras que en la siguiente sección se cubren los circuitos TTL en detalle. Aunque TTL es la familia lógica bipolar más comúnmente utilizada, ha sido sustituida en gran parte por las familias CMOS que hemos estudiado en las secciones anteriores.

Con todo, es útil estudiar el funcionamiento básico TTL para la ocasional aplicación que requiere de interfaces TTL/CMOS, como se discutió en la sección 3.12. También, un mejor entendimiento de TTL puede darnos idea de la similitud fortuita de niveles lógicos que permitieron a la industria emigrar suavemente desde la lógica TTL a la lógica CMOS de 5 V, y ahora a la lógica CMOS de menor voltaje y mayor rendimiento de 3.3 V, como se describe en la sección 3.13. Si usted no está interesado en todos los sangrientos detalles de TTL, puede saltarse a la sección 3.11, donde hay un resumen de las familias TTL.

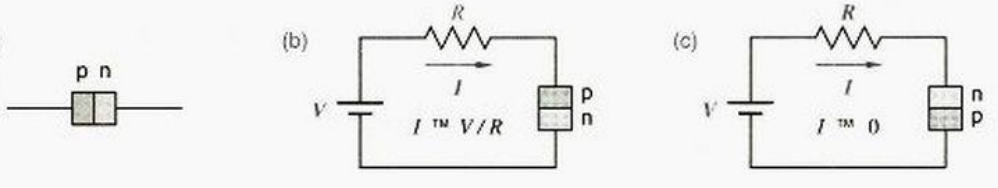
3.9.1 Diodos

Un *diodo semiconductor* se fabrica de dos tipos de material semiconductor, llamados tipo p y tipo n, que se ponen en contacto entre sí como se ilustra en la figura 3-61(a). Éste es básicamente el mismo material que se utiliza en los transistores MOS de canal-p y canal-n. El punto de contacto entre los materiales p y n se denomina una *unión pn*. (En realidad, un diodo se fabrica normalmente a partir de un cristal monolítico de material semiconductor en el cual las dos mitades son "dopadas" o se les agregan diferentes impurezas para darles propiedades de tipo p y de tipo n.)

diodo semiconductor
material de tipo-p
material de tipo-n
unión pn

Las propiedades físicas de una unión pn son tales que la corriente positiva puede fluir fácilmente desde el material tipo p hacia el material tipo n. De este modo, si construimos el circuito mostrado en la figura 3-61(b), la unión pn actúa casi como un cortocircuito. Sin embargo, las propiedades físicas también hacen muy difícil que la corriente

Figura 3-61 Diodos semiconductores: a) la unión pn; b) unión directamente polarizada que permite el flujo de corriente; c) unión inversamente polarizada que bloquean el flujo de corriente.



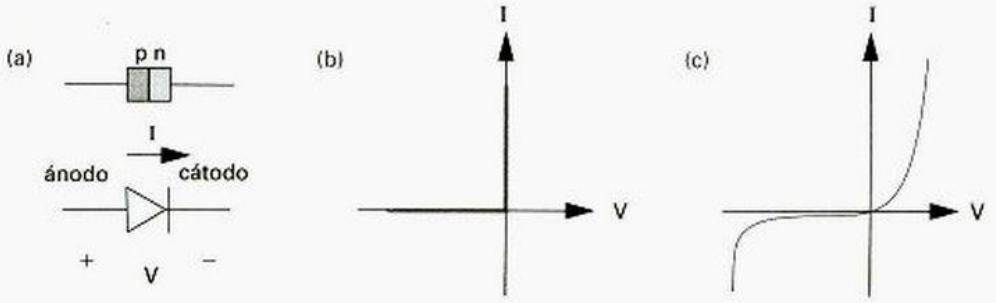


Figura 3-62 Diodos: a) símbolo; b) característica de transferencia de un diodo ideal; c) característica de transferencia de un diodo real.

acción de diodo

positiva fluya en la dirección opuesta, de *n* a *p*. Así, en el circuito de la figura 3-61(c), la unión *pn* se comporta casi como un circuito abierto. Esto se denomina *acción del diodo*.

diodo

cátodo

Aunque es posible construir tubos de vacío y otros dispositivos que exhiban acción de diodo, los sistemas modernos hacen uso de uniones *pn* (diodos semiconductores), a los cuales, de ahora en adelante, llamaremos simplemente *diodos*. La figura 3-62(a) muestra el símbolo esquemático de un diodo. Como lo hemos mostrado, en operación normal cantidades significativas de corriente pueden fluir solamente en la dirección indicada por las dos flechas, del *ánodo* al *cátodo*. En efecto, el diodo actúa como un cortocircuito mientras que el voltaje que cruza la unión ánodo-cátodo no es negativo. Si el voltaje del ánodo a cátodo es negativo, el diodo actúa como un circuito abierto y no fluye corriente.

diodo inversamente polarizado

diodo directamente polarizado

La característica de transferencia de un diodo ideal que se muestra en la figura 3-62(b) ilustra adicionalmente este principio. Si el voltaje de ánodo a cátodo, *V*, es negativo, se dice que el diodo se encuentra *inversamente polarizado* y la corriente *I* a través del diodo es cero. Si *V* no es negativo, se dice que el diodo está *polarizado directamente* y la *I* puede ser un valor positivo arbitrariamente grande. De hecho, *V* nunca puede llegar a ser más grande que cero, porque un diodo ideal actúa como un cortocircuito de resistencia cero cuando se polariza directamente.

Un diodo real, no ideal, tiene una resistencia que es menos de infinito cuando se polariza inversamente, y mayor que cero cuando se polariza directamente, de modo que las características de transferencia se ven como en la figura 3-62(c). Cuando se polalriza directamente, el diodo actúa como una pequeña resistencia no lineal; su caída de voltaje

SÍ, HAY DOS FLECHAS

... en la figura 3-62(a). La segunda flecha está integrada en el símbolo del diodo para ayudarle a recordar la dirección del flujo de corriente. Una vez que se sabe esto, hay muchas formas de recordar cuál extremo se denomina ánodo y cuál es el cátodo. Los aficionados de los amplificadores de alta fidelidad de tubos de vacío podrán recordar que los electrones viajan del cátodo caliente hacia el ánodo, y por lo tanto el flujo de corriente positiva va desde el ánodo hacia el cátodo. Aquellos de nosotros que aún veíamos "Plaza Sésamo" cuando la mayor parte de los tubos de vacío quedaron descontinuados podemos pensar en términos del alfabeto: la corriente fluye alfabéticamente desde A hasta C. (Mi agradecimiento al fanático del pájaro Abelardo, Rick Casey, por sugerir este truco nemotécnico.)

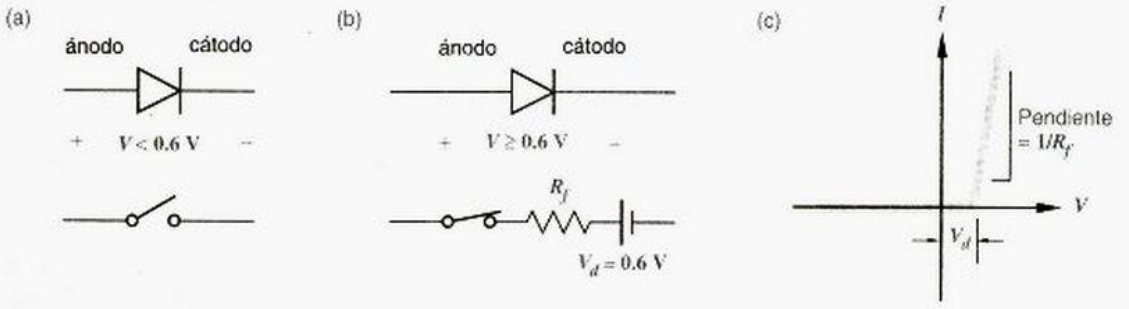


Figura 3-63 Modelo de un diodo real: a) inversamente polarizado; b) directamente polarizado; c) característica de transferencia del diodo directamente polarizado.

se incrementa a medida que la corriente aumenta, pero no de forma estrictamente proporcional. Cuando el diodo se polariza inversamente, fluye una pequeña cantidad de *corriente de fuga* negativa. Si el voltaje se hace demasiado negativo, el diodo alcanza su *ruptura*, y pueden fluir grandes cantidades de corriente negativa; en la mayoría de las aplicaciones, se impide este tipo de operación.

corriente de fuga
ruptura del diodo

Un diodo real puede modelarse de manera muy simple como se muestra en la figura 3-63(a) y (b). Cuando el diodo se encuentra inversamente polarizado, actúa como un circuito abierto; ignoramos la corriente de fuga. Cuando el diodo se encuentra directamente polarizado, actúa como una pequeña resistencia, R_f, en serie con V_d, una pequeña fuente de voltaje. R_f se denomina la *resistencia directa* del diodo, y V_d se conoce como la *caída del diodo*.

resistencia directa
caída de diodo

Una selección cuidadosa de los valores para R_f y V_d producen una razonable aproximación lineal por segmentos a la característica de transferencia real del diodo, como en la figura 3-63(c). En un diodo típico de señal pequeña como el 1N914, la resistencia directa R_f es de aproximadamente 25 Ω y la caída de voltaje en el diodo V_d es de alrededor de 0.6 V.

A fin de obtener una idea de aproximación a los diodos, se debería recordar que un diodo real en realidad no contiene la fuente de 0.6 V que aparece en el modelo. Es sólo que, debido a la no linealidad de la característica de transferencia del diodo real, cantidades significativas de corriente no comienzan a fluir hasta que el voltaje V directo del diodo ha alcanzado aproximadamente 0.6 V. También nótese que en aplicaciones típicas, la resistencia directa de 25 Ω del diodo es pequeña en comparación con otras resistencias en el circuito, de modo que se presenta muy poca caída de voltaje adicional a través del diodo directamente polarizado una vez que V ha alcanzado los 0.6 V. De este modo, para propósitos prácticos, se considera que un diodo directamente polarizado tiene una caída fija de más o menos 0.6 V.

DIODOS ZENER Los *diodos Zener* se aprovechan de la ruptura de diodo, en particular de la abrupta pendiente *V-I* en la región de ruptura. Un diodo Zener puede funcionar como un regulador de voltaje cuando se utiliza con un resistor para limitar la corriente de ruptura. Una amplia variedad de Zeners con diferentes voltajes de ruptura son producidos para aplicaciones de regulador de voltaje.

Tabla 3-9
Niveles lógicos en un sistema lógico de diodo simple.

Nivel de señal	*Denominación*	*Valor lógico binario*
0–2 volts	BAJO	0
2–3 volts	margen de ruido	indefinido
3–5 volts	ALTO	1

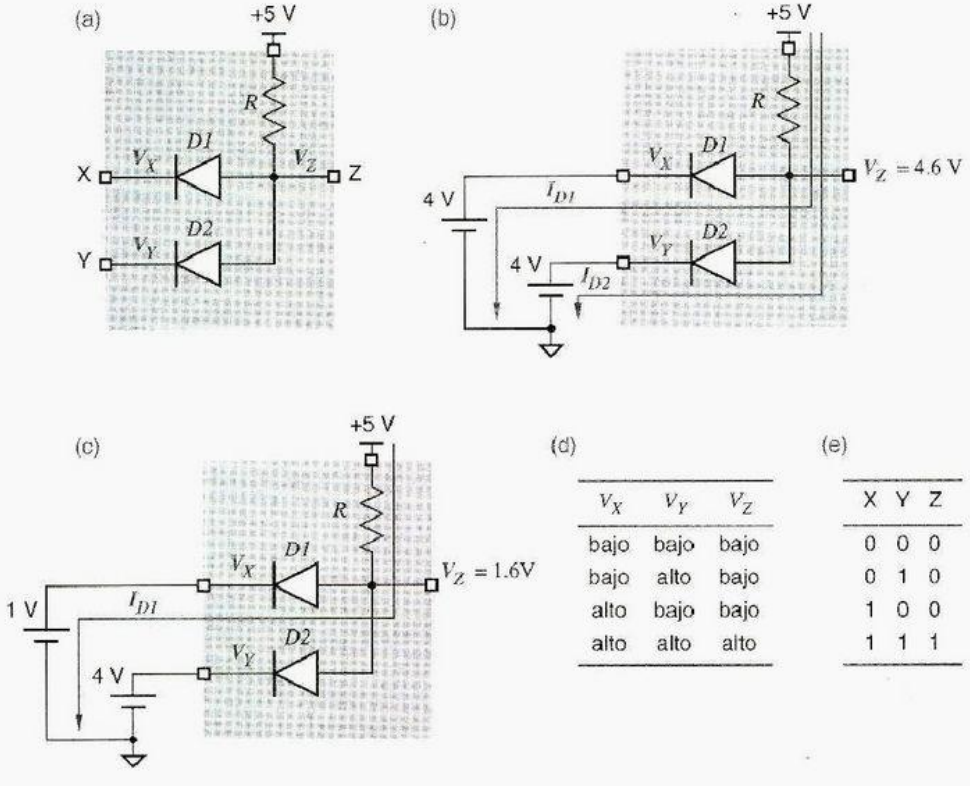
3.9.2 Lógica de diodos

BAJO
ALTO

La acción del diodo es explotada para realizar operaciones lógicas. Considere un sistema lógico con una alimentación de 5 V y las características mostradas en la tabla 3-9. Dentro del intervalo de 5 volts, los voltajes de señal están divididos en dos intervalos, BAJO y ALTO, con un margen de ruido de 1 volt entre ellos. Un voltaje en el intervalo BAJO se considera como un 0 lógico y un voltaje en el intervalo ALTO es un 1 lógico.

compuerta AND con diodos

Con estas definiciones, una *compuerta AND de diodo* puede construirse como se muestra en la figura 3-64(a). En este circuito, supongamos que ambas entradas X y Y se encuentran conectadas a fuentes de voltaje ALTO, digamos 4 V, de manera que *VX* y *VY* sean ambos de 4 V como en (b). Entonces ambos diodos se encuentran directamente polarizados y el voltaje de salida *VZ* es una caída de diodo arriba de 4 V, o arriba de 4.6 V. Una pequeña cantidad de corriente, determinada por el valor de *R*, fluye desde la alimentación de 5 V a través de los dos diodos y en las fuentes de 4 V. Las flechas tenues en la figura muestran la trayectoria de este flujo de corriente.



(d)

V_X	V_Y	V_Z
bajo	bajo	bajo
bajo	alto	bajo
alto	bajo	bajo
alto	alto	alto

(e)

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Figura 3-64
Una compuerta AND con diodos: a) circuito eléctrico; b) ambas entradas en ALTO; c) una entrada en ALTO, una en BAJO; d) tabla de funciones; e) tabla de verdad.

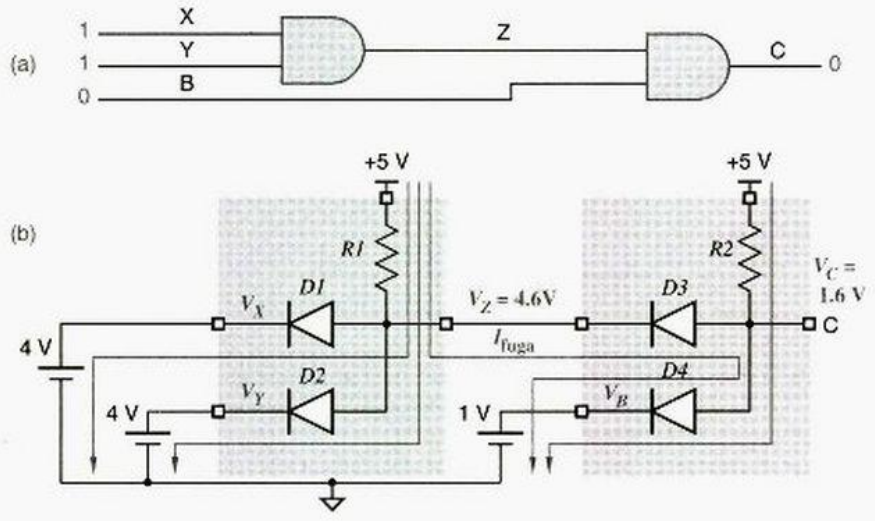


Figura 3-65
Dos compuertas AND: a) diagrama lógico; b) circuito eléctrico.

Ahora supongamos que *VX* decae a 1 V como en la figura 3-64(c). En la compuerta AND de diodo, el voltaje de salida es igual al más bajo de los dos voltajes de entrada más una caída del diodo. De este modo, *VZ* decae a 1.6 V y el diodo *D2* se polariza inversamente (el ánodo se encuentra a 1.6 V y el cátodo todavía está a 4 V). La entrada BAJO sola "disminuye" la salida de la compuerta AND de diodo a un valor BAJO. Obviamente, dos entradas BAJO crean una salida BAJO también. Esta operación funcional se resume en (d) y se repite en términos de valores de lógica binaria en (e); claramente, ésta es una compuerta AND.

La figura 3-65(a) ilustra un circuito lógico con dos compuertas AND conectadas juntas; la figura 3-65(b) muestra el circuito eléctrico equivalente con un conjunto particular de valores de entrada. Este ejemplo muestra la necesidad de diodos en el circuito AND: *D3* permite que la salida Z de la primera compuerta AND permanezca en nivel ALTO mientras que la salida C de la segunda compuerta AND se empuja al nivel BAJO por la entrada B a través *D4*.

Cuando las compuertas lógicas de diodos se establecen en cascada como en la figura 3-65, los niveles de voltaje de las señales lógicas se alejan de las líneas de alimentación y tienden hacia la región indefinida. De esta manera, en la práctica, una compuerta AND de diodo normalmente debe estar seguida por un amplificador de transistor para restablecer los niveles lógicos; éste es el esquema empleado en las compuertas NAND TTL, descritas en la sección 3.10.1. Sin embargo, los diseñadores lógicos se encuentran tentados ocasionalmente a utilizar diodos discretos para realizar la lógica bajo circunstancias especiales; véase por ejemplo, el ejercicio 3.95.

3.9.3 Transistores bipolares de unión

Un *transistor bipolar de unión* es un dispositivo de tres terminales que, en la mayoría de los circuitos lógicos, actúa como un interruptor controlado por corriente. Si colocamos una pequeña corriente en una de las terminales, denominada la *base*, entonces el interruptor se encuentra "encendido" (la corriente puede fluir entre las otras dos terminales, llamadas el *emisor* y el *colector*). Si no hay corriente en la base, entonces el interruptor se encontrará "apagado", es decir, no fluirá corriente alguna entre el emisor y el colector.

transistor bipolar de unión

base

emisor

colector

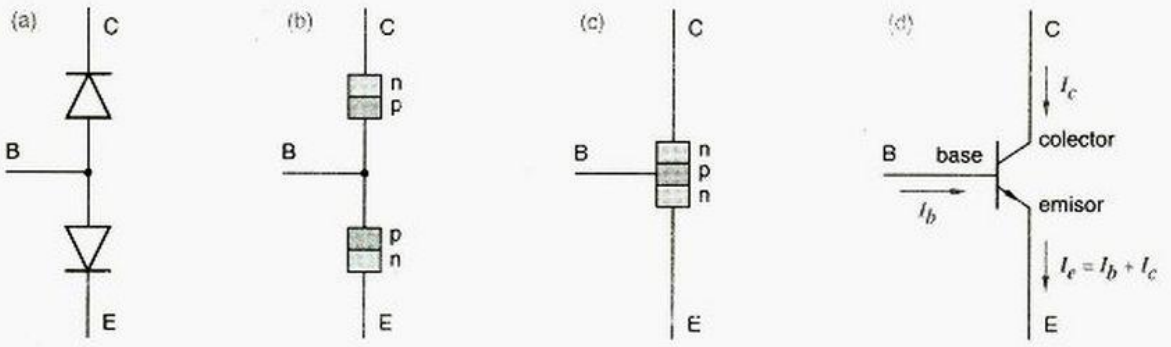


Figura 3-66 Desarrollo de un transistor *npn*: a) diodos espalda con espalda; b) uniones *pn* equivalentes; c) estructura de un transistor *npn*; d) símbolo del transistor *npn*.

Para estudiar el funcionamiento de un transistor, primero consideremos la operación de un par de diodos conectados como se ilustra en la figura 3-66(a). En este circuito, la corriente puede fluir desde el nodo B hasta el nodo C o el nodo E, cuando el diodo apropiado se polariza en forma directa. Sin embargo, no puede fluir corriente desde C a E, o viceversa, puesto que para cualquier elección de voltajes en los nodos B, C y E, uno o ambos diodos estarán inversamente polarizados. Las uniones *pn* de los dos diodos en este circuito se muestran en (b).

transistor npn

Ahora supongamos que fabricamos los diodos espalda con espalda de modo que ellos compartan una región tipo *p* común, como se ilustra en la figura 3-66(c). La estructura resultante se denomina un *transistor npn* y tiene una propiedad sorprendente. (¡Por lo menos los físicos que trabajaban con transistores allá por 1950 pensaban que era sorprendente!) Si hacemos pasar una corriente a través de la unión *pn* base a emisor, entonces la corriente también es habilitada para fluir a través de la unión *np* de colector a base (lo cual normalmente es imposible) y desde allí hacia el emisor.

El símbolo del circuito para el transistor *npn* se muestra en la figura 3-66(d). Nótese que el símbolo contiene una diminuta flecha en dirección del flujo de corriente positivo. Esto también nos recuerda que la unión base a emisor es una unión *pn*, lo mismo que un diodo cuyo símbolo tiene una flecha que apunta en la misma dirección.

transistor pnp

También es posible fabricar un *transistor pnp*, como se ilustra en la figura 3-67. Sin embargo, los transistores *pnp* rara vez son utilizados en circuitos digitales, de modo que no los discutiremos en adelante.

amplificador
región activa

La corriente I_e que fluye fuera del emisor de un *transistor npn* es la suma de las corrientes I_b e I_c que fluyen hacia la base y el colector. Un transistor se utiliza frecuentemente como un *amplificador* de señales, porque sobre cierto intervalo de operación (la *región activa*) la corriente de colector es igual a una constante fija por la corriente de base ($I_c = \beta \cdot I_b$). Sin embargo, en circuitos digitales, normalmente utilizamos un transistor como un interruptor simple que siempre se encuentra completamente "encendido" o completamente "apagado", como se explica a continuación.

configuración de emisor común

La figura 3-68 muestra la *configuración de emisor común* de un transistor *npn*, que es la más frecuentemente utilizada en aplicaciones de conmutación digital. Esta configuración

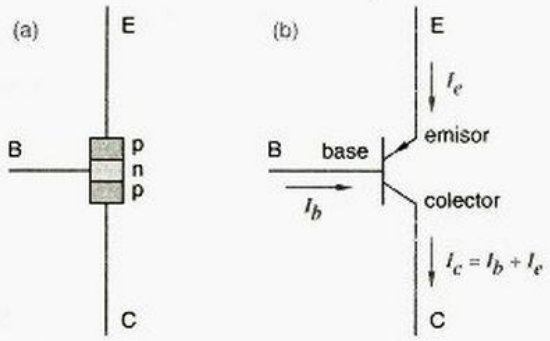


Figura 3-67
Un transistor *pnp*:
a) estructura; b) símbolo.

utiliza dos resistores discretos, *R1* y *R2*, además de un transistor *npn* simple. En este circuito, si V_{ENT} es 0 o negativo, entonces la unión de diodo base a emisor se encuentra inversamente polarizada, y no podrá fluir corriente de base (I_b). Si no fluye la corriente de base, entonces no podrá fluir corriente de colector (I_c) y se dice que el transistor está en *corte* (*OFF*).

corte (apagado, OFF)

Puesto que la unión base a emisor es un diodo *real*, en oposición a uno ideal, V_{ENT} debe alcanzar por lo menos +0.6 V (una caída de diodo) antes de que cualquier corriente de base pueda fluir. Una vez que esto ocurre, la ley de Ohm nos dice que

$$I_b = (V_{ENT} - 0.6) / R1$$

(Ignoramos la resistencia directa R_f de la unión base a emisor directamente polarizada, la cual por lo regular es pequeña en comparación con el resistor de base *R1*.) Cuando fluye la corriente de base, entonces la corriente de colector puede fluir en una cantidad proporcional a I_b, es decir,

$$I_c = \beta \cdot I_b$$

La constante de proporcionalidad, β, se conoce como la *ganancia* del transistor, y se encuentra en el intervalo de 10 a 100 para los transistores típicos.

β *(beta)*
ganancia

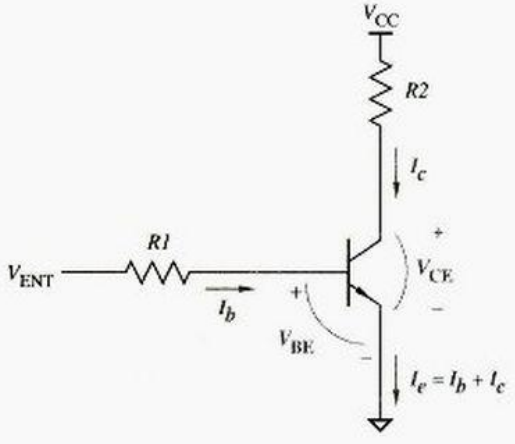


Figura 3-68
Configuración de emisor común de un transistor *npn*.

Aunque la corriente de base I_b controla el flujo de corriente del colector I_c, también controla indirectamente el voltaje V_{CE} a través de la unión colector a emisor, puesto que V_{CE} es precisamente el voltaje de alimentación V_{CC} menos la caída de voltaje a través del resistor *R2*:

$$\begin{aligned} V_{CE} &= V_{CC} - I_c \cdot R2 \\ &= V_{CC} - \beta \cdot I_b \cdot R2 \\ &= V_{CC} - (V_{ENT} - 0.6) \cdot R2 / R1 \end{aligned}$$

Sin embargo, en un transistor ideal V_{CE} nunca puede ser menor que cero (el transistor no puede crear un potencial negativo), y en un transistor real V_{CE} nunca puede ser menor que $V_{CE(sat)}$, un parámetro del transistor que se encuentra por lo regular alrededor de 0.2 V.

Si los valores de V_{ENT}, β, *R1* y *R2* son de tal modo que la ecuación anterior prediga un valor de V_{CE} que sea menor a $V_{CE(sat)}$, entonces el transistor no puede estar funcionando en la región activa y la ecuación no se aplica. En vez de ello, el transistor se encuentra funcionando en la *región de saturación*, y se dice que está *saturado (ON)*. No importa cuánta corriente I_b suministremos a la base, V_{CE} no puede caer por debajo de $V_{CE(sat)}$, y la corriente de colector I_c estará determinada principalmente por el resistor de carga *R2*:

región de saturación

saturado (encendido, ON)

$$I_c = (V_{CC} - V_{CE(sat)}) / (R2 + R_{CE(sat)})$$

resistencia de saturación

Aquí, $R_{CE(sat)}$ es la *resistencia de saturación* del transistor. Por lo regular, $R_{CE(sat)}$ es de 50 Ω o menor y es insignificante en comparación con *R2*.

simulación de transistor

Los especialistas en computación pueden querer imaginarse un transistor *npn* como un dispositivo que continuamente examina su entorno y ejecuta el programa en la tabla 3-10.

3.9.4 Inversor en lógica de transistor

La figura 3-69 muestra que podemos hacer un inversor lógico a partir de un transistor *npn* en la configuración de emisor común. Cuando el voltaje de entrada es **BAJO**, el voltaje de salida es **ALTO**, y viceversa.

En las aplicaciones de conmutación digital, los transistores bipolares con frecuencia se hacen funcionar de modo que siempre estén en corte o saturados. Es decir, los circuitos

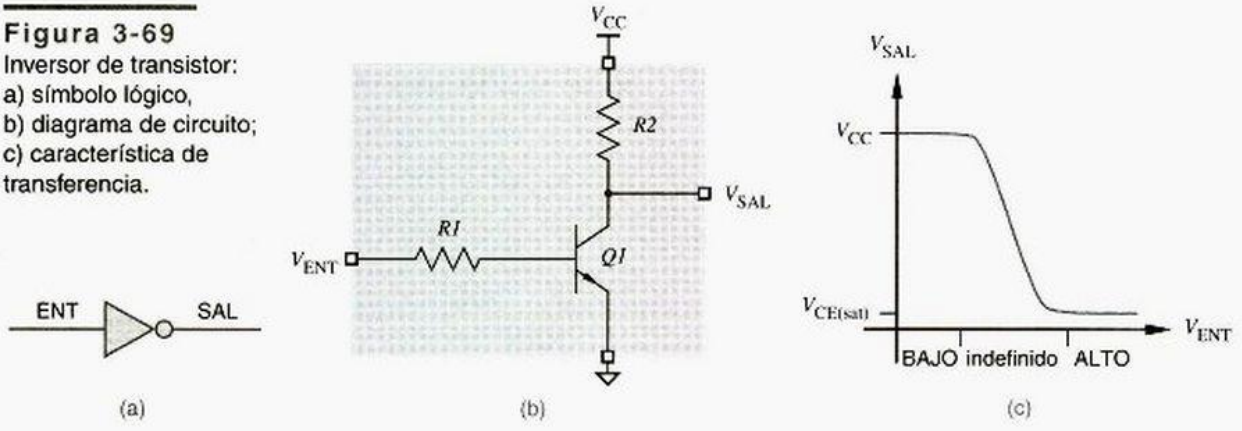


Figura 3-69
Inversor de transistor: a) símbolo lógico, b) diagrama de circuito; c) característica de transferencia.

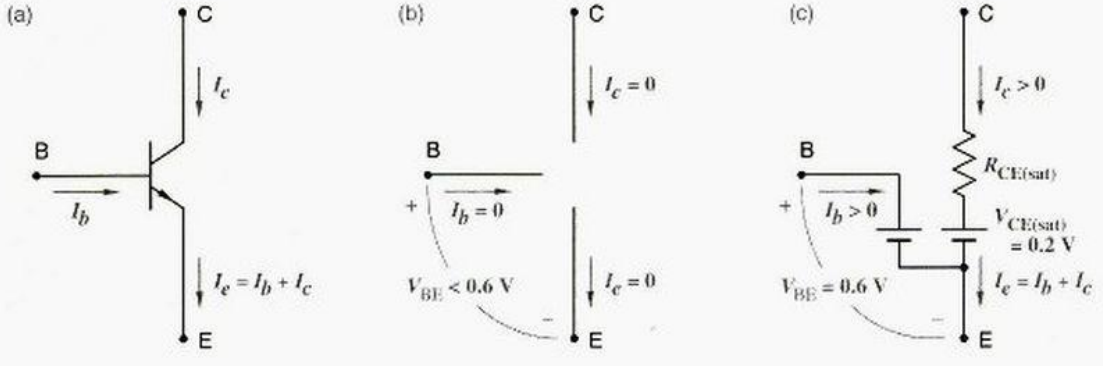
Tabla 3-10 Un programa en C que simula la función de un transistor *npn* en la configuración de emisor común.

```
/* transistor Parameters */
#define DIODEDROP 0.6 /* volts */
#define BETA 10;
#define VCE_SAT 0.2   /* volts */
#define RCE_SAT 50    /* ohms  */

main()
{
    float Vcc, Vin, R1, R2;  /*  circuit parameters */
    float Ib, Ic, Vce;       /*  circuit conditions  */

    if (Vin < DIODEDROP) {   /* cut off */
       Ib  = 0.0;
       Ic  = 0.0;
       Vce = Vcc;
    }
    else {                   /* active or saturated*/
        Ib = (Vin - DIODEDROP) / R1;
        if ((Vcc - ((BETA * Ib) * R2)) >= VCE_SAT) {   /* active */
            Ic  = BETA * Ib;
            Vce = Vcc - (Ic * R2);
        }
        else {               /* saturated */
           Vce = VCE_SAT;
           Ic  = (Vcc - Vce) / (R2 + RCE_SAT);
        }
    }
}
```

Figura 3-70 Estados normales de un transistor *npn* en un circuito de conmutación digital: a) corrientes y símbolo del transistor; b) circuito equivalente para un transistor en corte (apagado, OFF); c) circuito equivalente para un transistor saturado (encendido, ON).



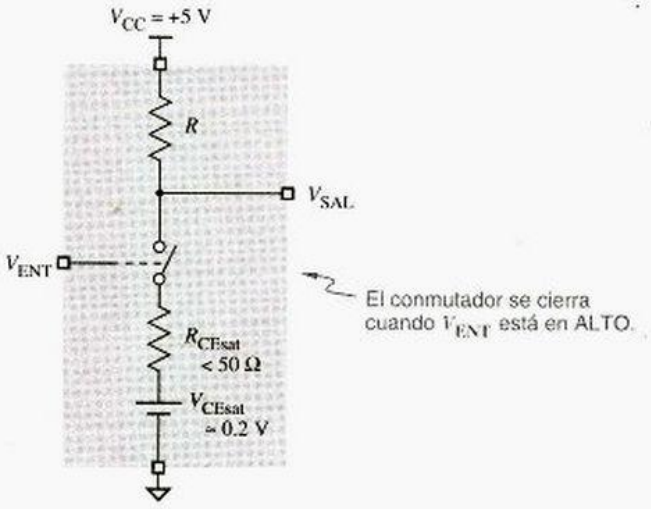


Figura 3-71
Modelo de conmutación para un inversor de transistor.

digitales tales como el inversor de la figura 3-69 están diseñados de modo que sus transistores se encuentren siempre (bueno, casi siempre) en uno de los estados descritos en la figura 3-70. Cuando el voltaje de entrada V_{ENT} está en nivel BAJO, es tan bajo como para que I_b sea cero y el transistor esté en corte; la unión de colector-emisor parece como un circuito abierto. Cuando V_{ENT} está en ALTO, es lo suficientemente alto (y $R1$ es lo suficientemente bajo y β está bastante alto) para que el transistor llegue a saturarse para cualquier valor razonable de $R2$; la unión de colector-emisor parecerá casi como un cortocircuito. Los voltajes de entrada en la región indefinida entre BAJO y ALTO no se permiten, excepto durante transiciones. Esta región indefinida corresponde al margen de ruido que discutimos en conjunto con la tabla 3-1.

Otra manera de visualizar la operación de un inversor de transistor se ilustra en la figura 3-71. Cuando V_{ENT} se encuentra en ALTO, el interruptor de transistor está cerrado, y la terminal de salida está conectada a tierra, sin duda un voltaje BAJO. Cuando V_{ENT} se encuentra en BAJO, el interruptor de transistor está abierto y la terminal de salida se lleva a +5 V a través de un resistor; el voltaje de salida es ALTO a menos que la terminal de salida se encuentre demasiado cargada (es decir, conectada de manera impropia a través de una baja impedancia hacia tierra).

3.9.5 Transistores Schottky

tiempo de almacenamiento

Cuando la entrada de un transistor saturado cambia, la salida no se modifica de inmediato; le toma un tiempo extra, denominado *tiempo de almacenamiento*, salir de saturación. De hecho, el tiempo de almacenamiento explica una parte significativa del retardo de propagación en la familia lógica TTL original.

diodo Schottky

transistor fijado Schottky
transistor Schottky

El tiempo de almacenamiento puede ser eliminado y el retardo de propagación puede reducirse asegurando que los transistores no se saturarán en operación normal. Las familias lógicas TTL contemporáneas hacen esto colocando un *diodo Schottky* entre la base y el colector de cada transistor que podría saturarse, como se muestra en la figura 3-72. Los transistores resultantes, que no se saturan, se conocen como *transistores* fijados *Schottky* o, para abreviar *transistores Schottky*.

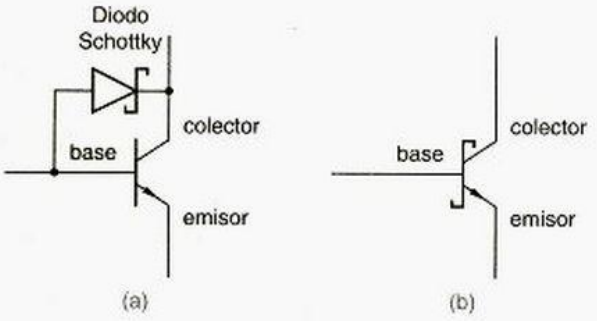


Figura 3-72
Transistor enclavado Schottky: a) circuito; b) símbolo.

Cuando está polarizado directamente, la caída de voltaje en el diodo Schottky es mucho menor que la de un diodo estándar, 0.25 V en comparación con 0.6 V. En un transistor saturado estándar, el voltaje de base a colector es de 0.4 V, como se ilustra en la figura 3-73(a). En un transistor Schottky, el diodo Schottky deriva la corriente de la base al colector antes que el transistor vaya a la saturación, como se muestra en (b). La figura 3-74 muestra el diagrama del circuito de un inversor sencillo que utiliza un transistor Schottky.

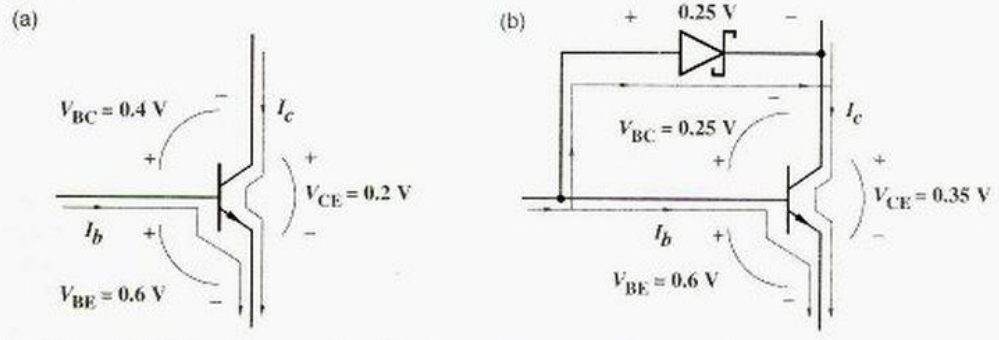


Figura 3-73 Funcionamiento de un transistor con corriente de base grande: a) transistor estándar saturado; b) transistor con diodo Schottky para evitar la saturación.

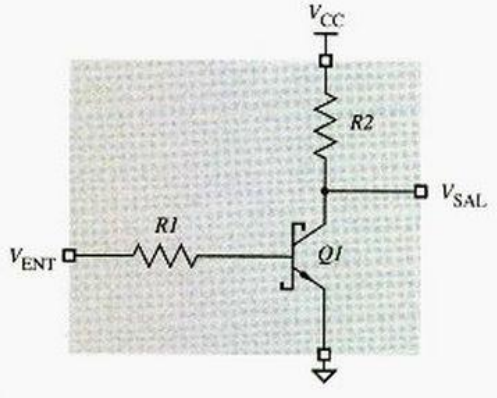


Figura 3-74
Inversor utilizando transistor Schottky.

3.10 Lógica transistor-transistor

La familia lógica bipolar más comúnmente utilizada es la lógica transistor-transistor. En realidad, existen muchas familias TTL diferentes, con un intervalo de velocidad, consumo de energía y otras características. Los ejemplos de circuitos en esta sección están basados en una familia TTL representativa, la denominada Schottky de baja potencia (LS o LS-TTL, "Low-power Schottky").

Las familias TTL utilizan básicamente los mismos niveles lógicos que las familias CMOS compatibles con TTL de las secciones anteriores. Utilizaremos las definiciones siguientes de niveles BAJO y ALTO en nuestras discusiones acerca del comportamiento de circuitos TTL:

BAJO 0–0.8 volts.
ALTO 2.0–5.0 volts.

3.10.1 Compuerta NAND TTL básica

El diagrama de circuito de una compuerta NAND LS-TTL de dos entradas, número de parte 74LS00, se muestra en la figura 3-75. La función NAND se obtiene al combinar una compuerta AND de diodo con un buffer inversor. El funcionamiento del circuito se comprende mejor al dividirlo en tres partes que se ilustran en la figura y se discuten en los tres párrafos siguientes:

- Compuerta AND de diodo y protección de entrada.
- Divisor de fase.
- Etapa de salida.

compuerta AND de diodo
diodo fijado

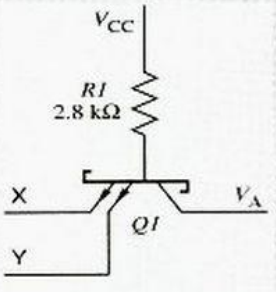
Los diodos *D1X* y *D1Y* y el resistor *R1* en la figura 3-75 forman una *compuerta AND de diodo*, como en la sección 3.9.2. Los *diodos de fijación D2X* y *D2Y* no hacen nada en operación normal, pero limitan las excursiones negativas no deseables en las entradas para una caída de diodo simple. Tales excursiones negativas pueden ocurrir en las transiciones de entrada de ALTO a BAJO como resultado de efectos de línea de transmisión, discutidos en la sección 11.4.

divisor de fase

En transistor *Q2* y los resistores que le rodean forman un *divisor de fase* que controla la etapa de salida. Dependiendo si la compuerta AND de diodo produce un voltaje "bajo" o "alto" en V_A, *Q2* está en corte o encendido.

¿QUÉ LUGAR OCUPA EN EL MUNDO *Q1*?

Nótese que no hay transistor *Q1* en la figura 3-75, pero los otros transistores se nombran de una manera ya tradicional; algunos dispositivos TTL de hecho tienen un transistor llamado *Q1*. En lugar de diodos como *D1X* y *D1Y*, estos dispositivos utilizan un transistor de emisor múltiple *Q1* para realizar la lógica. Este transistor tiene un emisor por entrada lógica, como se ilustra en la figura a la derecha. Llevando cualquiera de los emisores hasta nivel BAJO es suficiente para encender el transistor y así llevar a V_A al nivel BAJO.



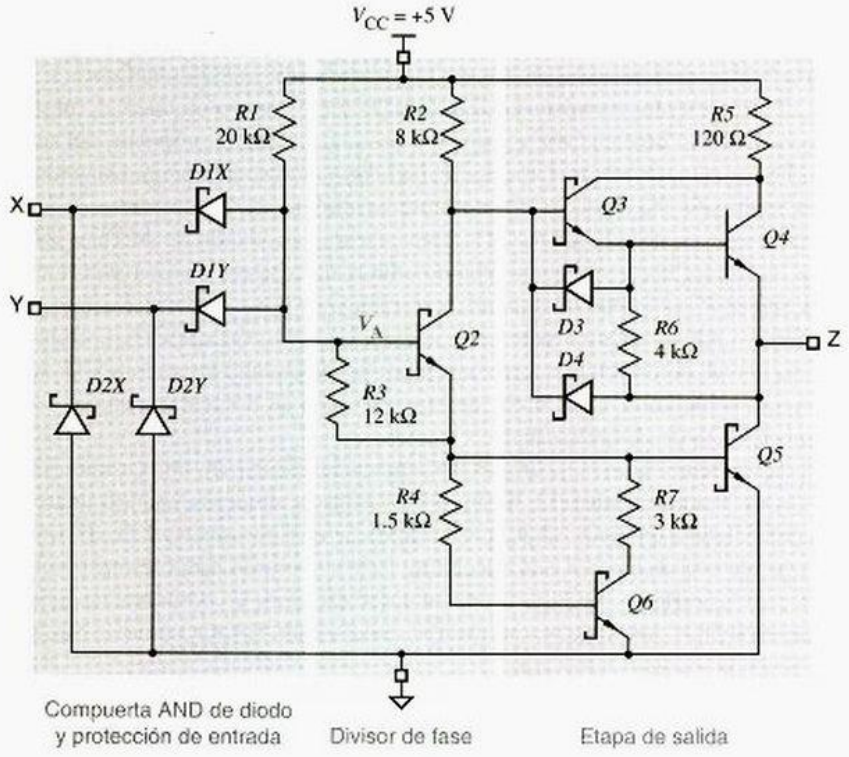


Figura 3-75
Diagrama de circuito de compuerta NAND LS-TTL de dos entradas.

La *etapa de salida* tiene dos transistores, *Q4* y *Q5*, uno de los cuales se encuentra encendido en cualquier momento. La etapa de salida TTL se denomina en ocasiones *salida totem-pole* o *push-pull*. De manera semejante a los transistores de canal-*p* y canal-*n* en CMOS, *Q4* y *Q5* proporcionan arranque y disminución activos a los estados ALTO y BAJO, respectivamente.

etapa de salida

salida totem-pole o salida push-pull

La operación funcional de la compuerta NAND TTL se resume en la figura 3-76(a). La compuerta efectúa realmente la función NAND, con la tabla de verdad y el símbolo lógico que se muestra en (b) y (c). Las compuertas NAND TTL pueden diseñarse con cualquier número deseado de entradas simplemente cambiando el número de diodos en

(a)

X	Y	V_A	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	V_Z	Z
L	L	≤1.05	off	on	on	off	off	2.7	H
L	H	≤1.05	off	on	on	off	off	2.7	H
H	L	≤1.05	off	on	on	off	off	2.7	H
H	H	1.2	on	off	off	on	on	≤0.35	L

(b)

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

(c)

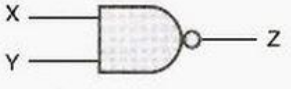


Figura 3-76
Operación funcional de una compuerta NAND de dos entradas TTL: a) tabla de funciones; b) tabla de verdad; c) símbolo lógico.

la compuerta AND de diodo de la figura. Las compuertas NAND TTL comercialmente disponibles tienen hasta 13 entradas. Un inversor TTL está diseñado como una compuerta NAND de 1 entrada, omitiendo los diodos *D1Y* y *D2Y* en la figura 3-75.

Puesto que los transistores de salida *Q4* y *Q5* son normalmente complementarios (uno en ON y el otro en OFF), usted tal vez se pregunte el propósito del resistor *R5* de 120 Ω en la etapa de salida. Un valor de 0 Ω daría incluso una mejor capacidad de control en el estado ALTO. Esto es ciertamente verdadero desde un punto de vista de CD. Sin embargo, cuando la salida TTL se encuentra cambiando de ALTO a BAJO o viceversa, existe un breve momento cuando ambos transistores pueden estar encendidos. El propósito de *R5* es limitar la cantidad de corriente que fluye desde VCC a tierra durante este momento. Incluso con un resistor de 120 Ω en la etapa de salida TTL, corrientes mayores que la normal, denominadas corrientes pico, fluyen cuando las salidas TTL se conmutan. Éstas son semejantes a los picos de corriente que se presentan cuando conmutan las salidas CMOS de alta velocidad.

Hasta ahora hemos mostrado las señales de salida para una compuerta TTL como fuentes de voltaje ideales. La figura 3-77 ilustra la situación cuando una entrada TTL está controlada a un nivel BAJO por la salida de otra compuerta TTL. El transistor *Q5A* en la compuerta de control está encendido (ON) y por ello proporciona una trayectoria a tierra para la corriente que fluye fuera del diodo *D1XB* en la compuerta de control. Cuando la corriente fluye *dentro* de una salida TTL en el estado BAJO, como en este caso, se dice que la salida *consume corriente*.

corriente consumida

La figura 3-78 muestra el mismo circuito con una salida ALTO. En este caso, *Q4A* en la compuerta de control se enciende lo suficiente para suministrar la pequeña cantidad de corriente de fuga fluyendo a través de los diodos inversamente polarizados *D1XB* y *D2XB* en la compuerta de control. Cuando la corriente fluye fuera de una salida TTL en el estado ALTO, se dice que la salida es una *fuente de corriente*.

corriente suministrada (de fuente)

3.10.2 Niveles lógicos y márgenes de ruido

Al principio de esta sección, indicamos que consideraríamos las señales TTL entre 0 y 0.8 V como de nivel BAJO y las señales entre 2.0 y 5.0 V como de nivel ALTO. En realidad, podemos ser más precisos al definir los niveles de entrada y salida TTL del mismo modo que lo hicimos para CMOS:

V_{OHmin} El voltaje de salida mínimo en el estado ALTO, 2.7 V para la mayoría de las familias TTL.

V_{IHmin} El voltaje de entrada mínimo garantizado para ser reconocido como un nivel ALTO, 2.0 V para todas las familias TTL.

PICOS DE CORRIENTE OTRA VEZ

Los picos de corriente pueden notarse como ruido en la fuente de alimentación y en conexiones de tierra en circuitos TTL y CMOS, especialmente cuando múltiples salidas se conmutan de manera simultánea. Por esta razón, los circuitos confiables requieren capacitores de desacoplamiento entre V_{CC} y tierra, distribuidos a lo largo del circuito de modo que hay un capacitor cada pulgada, o algo por el estilo, de cada chip. Los capacitores de desacoplamiento suministran la corriente instantánea necesaria durante las transiciones.

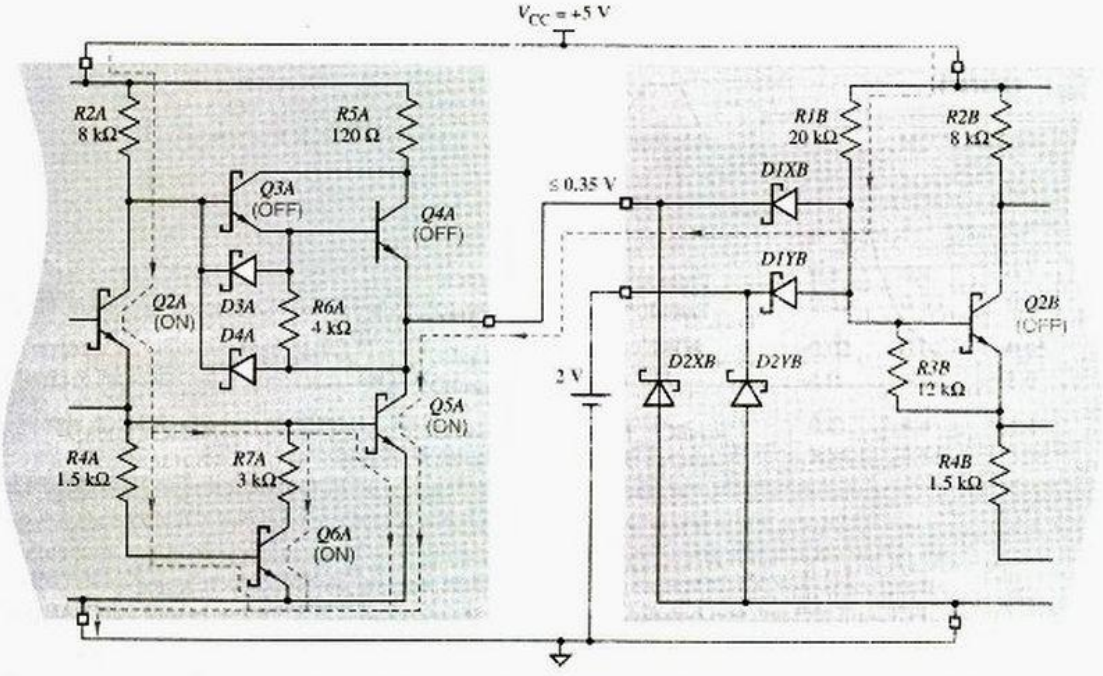


Figura 3-77 Una salida TTL que controla una entrada TTL en BAJO.

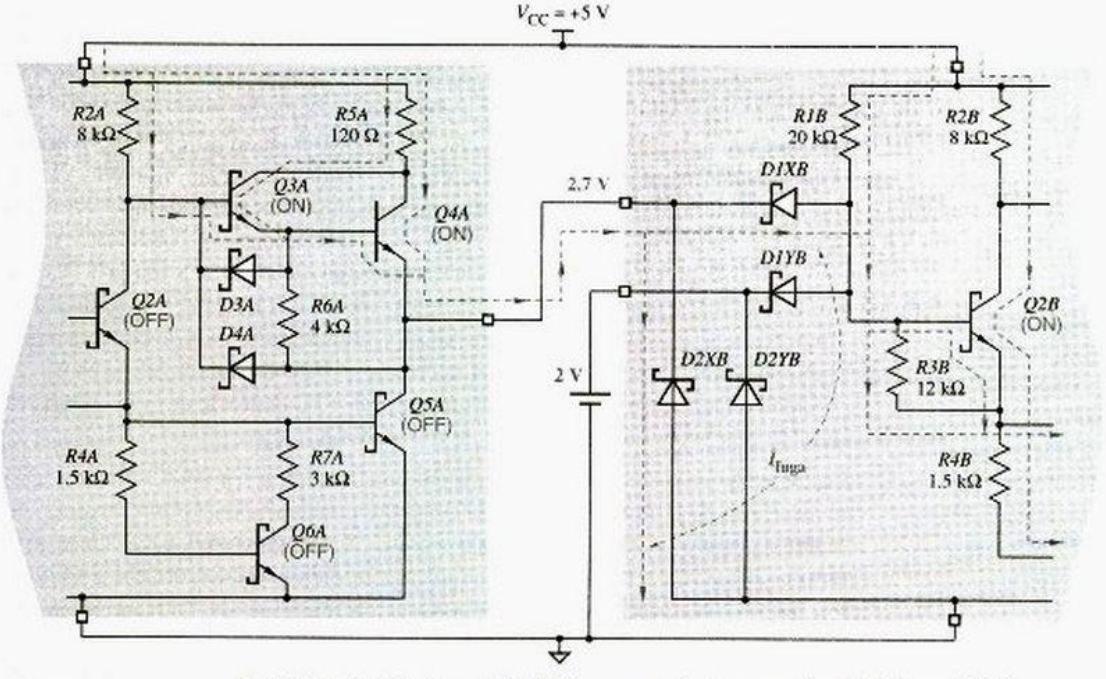


Figura 3-78 Una salida TTL que controla una entrada TTL en ALTO.

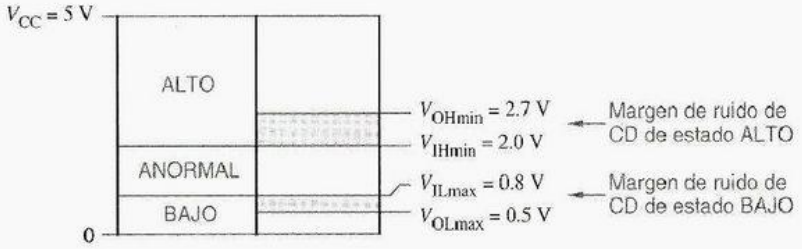


Figura 3-79
Márgenes de ruido para las familias lógicas TTL populares (74LS, 74S, 74ALS, 74AS, 74F).

V_{ILmax} El voltaje de entrada máximo garantizado para ser reconocido como un nivel BAJO, 0.8 V para la mayoría de las familias TTL:

V_{OLmax} El voltaje de salida máximo en el estado BAJO, y es de 0.5 V para la mayoría de las familias.

Estos márgenes de ruido se ilustran en la figura 3-79.

margen de ruido de CD

En el estado ALTO, la especificación V_{OHmin} de la mayoría de las familias TTL excede a V_{IHmin} por 0.7 V, de modo que TTL tiene un *margen de ruido de CD* de 0.7 V en el estado ALTO. Es decir, le toma por lo menos 0.7 V de ruido corromper una salida ALTO en el peor de los casos en un voltaje que no esté garantizado que sea reconocible como una entrada ALTO. Sin embargo, en el estado BAJO, V_{ILmax} excede V_{OLmax} por solamente 0.3 V, de modo que el margen de ruido de CD en el estado BAJO es de solamente 0.3 V. En general, los circuitos TTL y compatibles con TTL tienen la tendencia a ser más sensibles al ruido en el estado BAJO que en el estado ALTO.

3.10.3 Fan out

Fan out

Como lo definimos anteriormente en la sección 3.5.4, *fan out* es una medida del número de entradas de compuerta que se encuentran conectadas a (y controladas por) una sola salida de compuerta. Como demostramos en esa sección, el fan out CD de CMOS que controlan entradas CMOS es virtualmente ilimitado, porque las entradas CMOS casi no requieren corriente en cualquier estado, ALTO o BAJO. Éste no es el caso de las entradas TTL. Como resultado, existen límites muy definidos sobre fan out TTL o CMOS controlando entradas TTL, como aprenderemos en los párrafos que vienen a continuación.

flujo de corriente

Como en CMOS el *flujo de corriente* en una terminal de entrada o salida TTL está definida como positiva si la corriente en realidad fluye *dentro* de ésta, y negativa si la corriente fluye *fuera* de la terminal. Como resultado, cuando una salida se conecta a una o más entradas, la suma algebraica de todas las corrientes de entrada y salida es 0.

La cantidad de corriente requerida por una entrada TTL depende de si la entrada es ALTO o BAJO, y se especifica por medio de dos parámetros:

I_{ILmax} La corriente máxima que requiere una entrada para llevarla a BAJO. Recuerde de la discusión de la figura 3-77 que la corriente positiva se encuentra fluyendo en realidad desde V_{CC}, pasando por *R1B*, a través del diodo *D1XB*, hacia fuera de la terminal de entrada, pasando por el transistor de salida de control *Q5A* y a tierra.

Puesto que la corriente fluye hacia fuera de una entrada TTL en el estado BAJO, I_{ILmax} tiene un valor negativo. La mayoría de las entradas LS-TTL tienen I_{ILmax} = -0.4 mA, lo que se da en llamar, en ocasiones, una *carga unitaria de estado BAJO* para LS-TTL.

carga unitaria de estado BAJO

I_{IHmax} La corriente máxima que requiere una entrada para llevarla a ALTO. Como se muestra en la figura 3-78 de la página 159, la corriente positiva fluye desde V_{CC}, pasando a través de *R5A* y *Q4A* de la compuerta de control, y *dentro* de la entrada controlada, donde sale a tierra a través de los diodos inversamente polarizados *D1XB* y *D2XB*.

Puesto que la corriente fluye *dentro* de una entrada TTL en el estado ALTO, I_{IHmax} tiene un valor positivo. La mayoría de las entradas LS-TTL tienen $I_{IHmax} = 20\ \mu A$, lo cual en ocasiones se conoce como la *carga unitaria de estado ALTO* para LS-TTL.

carga unitaria de estado ALTO

Como con las salidas CMOS, las salidas TTL pueden suministrar (como fuente) o consumir (como sumidero) cierta cantidad de corriente dependiendo del estado, ALTO o BAJO:

I_{OLmax} La corriente máxima que una salida puede consumir en el estado BAJO mientras se mantiene un voltaje de salida no mayor que V_{OLmax}. Puesto que la corriente fluye en la salida, I_{OLmax} tiene un valor positivo, 8 mA para la mayoría de las salidas LS-TTL.

I_{OLmax} La corriente máxima que una salida puede suministrar en el estado ALTO mientras se mantiene un voltaje de salida no menor que V_{OHmin}. Puesto que la corriente fluye fuera de la salida, I_{OHmax} tiene un valor negativo, $-400\ \mu A$ para la mayoría de las salidas LS-TTL.

Nótese que el valor de I_{OLmax} para las salidas típicas LS-TTL es exactamente 20 veces el valor absoluto de I_{ILmax}. Como resultado, se dice que LS-TTL tiene un *fan out en estado BAJO* de 20, debido a que una salida puede controlar hasta 20 entradas en el estado BAJO. Similarmente, el valor absoluto de I_{OHmax} es exactamente 20 veces I_{IHmax}, de manera que se dice que LS-TTL tiene un *fan out en estado ALTO* de 20 también. El fan out *total* es el menor de los fan aut de estado BAJO y ALTO.

fan out en estado BAJO

fan out en estado ALTO

fan out en total

Cargar una salida TTL con más de lo que permite el fan out nominal tiene los mismos efectos perjudiciales que se describieron para los dispositivos CMOS en la sección 3.5.5 de la página 110. Es decir, los márgenes de ruido de CD pueden ser reducidos o eliminados, los retardos y tiempos de transición pueden incrementarse y el dispositivo puede llegar a sobrecalentarse.

SALIDA TTL ASIMÉTRICA

Aunque las divergencias numéricas de LS-TTL para los estados ALTO y BAJO son iguales, LS-TTL y otras familias TTL tiene una asimetría definida en la capacidad de control de corriente: una salida LS-TTL puede consumir 8 mA en el estado BAJO, pero puede suministrar solamente 400 μA en el estado ALTO.

Esta asimetría no es problema cuando las salidas TTL controlan otras salidas TTL, debido a que coincide con una asimetría correspondiente en los requerimientos de corriente de entrada TTL (I_{ILmax} es grande, mientras que I_{IHmax} es pequeña). Sin embargo, es una limitación cuando TTL se utiliza para controlar LED, relevadores, solenoides u otros dispositivos que requieren grandes cantidades de corriente, con frecuencia decenas de miliamperes. Los circuitos que utilizan estos dispositivos deben diseñarse de modo que la corriente fluya (y el dispositivo controlado se "encienda") cuando la salida TTL se encuentre en el estado BAJO, y que muy poca o ninguna corriente fluya en el estado ALTO. Se hacen compuertas especiales TTL de buffer controlador que pueden consumir hasta 60 mA en el estado BAJO, pero que todavía tienen una capacidad algo endeble de suministro de corriente en el estado ALTO (2.4 mA).

DEDOS QUEMADOS Si una salida TTL o CMOS es forzado a consumir mucho más que I_{OLmax}, el dispositivo puede ser dañado, especialmente si se permite que la corriente alta fluya por más de un segundo. Por ejemplo, supongamos que una salida TTL en el estado BAJO esté en cortocircuito directamente con la alimentación de 5 V. La resistencia de encendido ON, $R_{CE(sat)}$, del transistor *Q5* saturado en una etapa típica de salida TTL es menor de 10 Ω. De este modo, *Q5* debe disipar alrededor de $5^2/10$ o 2.5 watts. ¡No intente esto a menos que usted se encuentre preparado para afrontar las consecuencias! Esto es suficiente calor como para destruir el dispositivo (y quemar sus dedos) en un tiempo muy corto.

En general, deben realizarse dos cálculos para confirmar que una salida no está siendo sobrecargada:

Estado ALTO Los valores I_{IHmax} para todas las entradas controladas se suman. Esta suma debe ser menor o igual que el valor absoluto de I_{OHmax} para la salida de control.

Estado BAJO Los valores I_{ILmax} para todas las entradas controladas se suman. El valor absoluto de esta suma debe ser menor o igual a I_{OLmax} para la salida de control.

Por ejemplo, supongamos que usted diseña un sistema en el que cierta salida LS-TTL controla diez entradas de compuerta LS-TTL y tres S-TTL. En el estado ALTO, se requiere un total de $10 \cdot 20 + 3 \cdot 50\ \mu A = 350\ \mu A$. Esto está dentro de una capacidad de suministro de corriente de estado ALTO de salida de LS-TTL de 400 μA. Pero en el estado BAJO, se requiere de un total de $10 \cdot 0.4 + 3 \cdot 2.0\ mA = 10.0\ mA$. Esto va más allá de una capacidad de consumo de corriente de estado BAJO de salida LS-TTL de 8 mA, de modo que la salida se encuentra sobrecargada.

3.10.4 Entradas sin utilizar

Las entradas sin utilizar de las compuertas TTL pueden ser manipuladas de la misma manera que describimos para las compuertas CMOS en la sección 3.5.6 de la página 111. Es decir, las entradas sin utilizar pueden ser vinculadas a las utilizadas, o las entradas sin usar pueden ser llevadas a nivel ALTO o BAJO, según como sea apropiado para la función lógica.

El valor de resistencia de un resistor de arranque o de disminución es más crítico con las compuertas TTL que con las compuertas CMOS, porque las entradas TTL extraen significativamente más corriente, especialmente en el estado BAJO. Si la resistencia es demasiado grande, la caída de voltaje a través del resistor puede provocar un voltaje de entrada de compuerta más allá del intervalo normal BAJO o ALTO.

Por ejemplo, considere el resistor de disminución (pull-down) mostrado en la figura 3-80. El resistor de disminución debe consumir 0.4 mA de corriente de cada una de las entradas LS-TTL sin utilizar que controla. La caída de voltaje a través del resistor no debe ser más de 0.5 V a fin de tener un voltaje de entrada BAJO no peor del producido por una salida de compuerta normal. Si el resistor controla *n* entradas LS-TTL, entonces debemos tener

$$n \cdot 0.4\ mA \cdot R_{pd} < 0.5\ V$$

ENTRADAS TTL FLOTANTES Un análisis de la estructura de entrada TTL muestra que las entradas sin utilizar que se dejan sin conectar (o sea, *flotando*) se comportan como si tuvieran aplicado un voltaje ALTO: son llevadas al nivel ALTO por el resistor de base *R1* en la figura 3-75 de la página 157. Sin embargo, el levantamiento o arranque de *R1* es más débil que el de una salida TTL controlando la entrada. Como resultado, una pequeña cantidad de ruido de circuito, tal como el producido por otras compuertas cuando se conmutan, puede hacer que una entrada flotante se comporte de manera espúrea como si estuviera en BAJO. Por lo tanto, en aras de la confiabilidad, la entradas sin utilizar TTL debería vincularse a una fuente de voltaje estable en ALTO o BAJO.

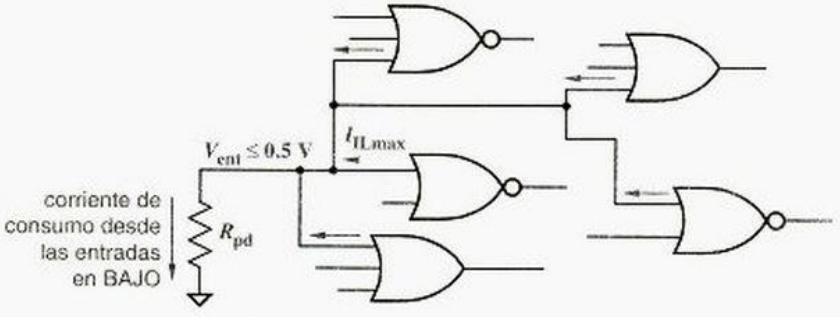


Figura 3-80 Resistor de disminución para entradas TTL.

De este modo, si el resistor debe llevar 10 entradas LS-TTL a nivel BAJO, entonces debemos tener $R_{pd} < 0.5 / (10 \cdot 4 \cdot 10^{-3})$ o $R_{pd} < 125\ \Omega$.

De manera semejante, considere el resistor de arranque mostrado en la figura 3-81. Debe suministrar 20 μA de corriente a cada entrada sin utilizar mientras que produce un voltaje ALTO no peor que el producido por una salida de compuerta normal, 2.7 V. Por lo tanto, la caída de voltaje a través del resistor no debe ser mayor de 2.3 V; si n entradas LS-TTL son controladas, debemos tener

$$n \cdot 20\ \mu A \cdot R_{pu} < 2.3\ \text{V}$$

De esta forma, si 10 entradas LS-TTL se activan, entonces $R_{pu} < 2.3 / (10 \cdot 20 \cdot 10^{-6})$ ·, o $R_{pu} < 11.5\ \text{K}\Omega$.

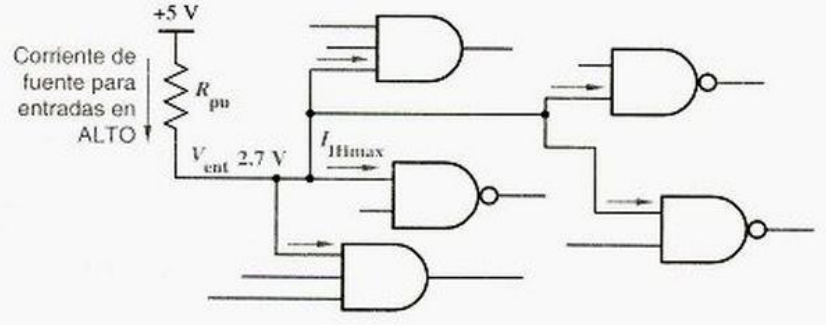


Figura 3-81 Resistor de arranque para entradas TTL.

¿POR QUÉ UTILIZAR UN RESISTOR?

Usted puede preguntarse, "¿por qué utilizar un resistor de arranque o de disminución, cuando una conexión directa a tierra o a la fuente de alimentación de 5 V debería ser una fuente perfectamente aceptable de nivel BAJO o ALTO?"

Bien, para una fuente de nivel ALTO, una conexión directa a la alimentación de 5 V no se recomienda, porque un transitorio de entrada arriba de 5.5 V puede dañar algunos dispositivos TTL, aquellos que emplean un transistor emisor múltiple en la etapa de entrada. El resistor de arranque limita la corriente y evita el daño en este caso.

Para una fuente de nivel BAJO, una conexión directa a tierra sin el resistor de disminución es en realidad correcta en la mayoría de los casos. Usted verá muchos ejemplos de esta clase de conexión a lo largo de este libro. Sin embargo, como se explica en la sección 11.2.2 de la página 904, el resistor de disminución todavía es deseable en algunos casos, de manera que la señal BAJO "constante" que produce pueda ser cancelada y llevada a ALTO para propósitos de prueba de sistemas.

3.10.5 Tipos adicionales de compuertas TTL

Aunque la compuerta NAND es el "caballo de batalla" de la familia TTL, pueden construirse otros tipos de compuertas con la misma estructura general de circuitos.

El diagrama de circuito para una compuerta NOR LS-TTL se muestra en la figura 3-82. Si cualquier entrada X o Y se encuentra en nivel ALTO, el transistor divisor de fase correspondiente *Q2X* o *Q2Y* se enciende, lo cual apaga *Q3* y *Q4* mientras activa *Q5* y *Q6*, y la salida es BAJO. Si ambas entradas están en BAJO, entonces ambos transistores divisores de fase se encuentran apagados, y la salida está forzada al nivel ALTO. Esta operación funcional se resume en la figura 3-83.

Los circuitos de entrada, divisor de fase y etapa de salida de la compuerta NOR LS-TTL son casi idénticos a los correspondientes de una compuerta NAND LS-TTL. La diferencia es que una compuerta NAND LS-TTL utiliza diodos para realizar la función AND, mientras que una compuerta NOR LS-TTL emplea transistores en paralelo en el divisor de fase para efectuar la función OR.

La velocidad, entrada y características de salida de una compuerta NOR TTL son comparables a las de una NAND TTL. Sin embargo, una compuerta NOR de n entradas emplea más transistores y resistores y es así más costosa en área de silicio que una compuerta NAND de n entradas. También, la corriente de fuga interna limita el número de transistores *Q2* que pueden ser situados en paralelo, de modo que las compuertas NOR tienen una fan in deficiente. (La compuerta NOR TTL discreta más grande tiene solamente 5 entradas, en comparación con una NAND de 13 entradas.) Como resultado, las compuertas NOR se utilizan con menos frecuencia que las compuertas NAND en diseños TTL.

Las compuertas TTL más "naturales" son las compuertas inversoras como las NAND y las NOR. las compuertas no inversoras TTL incluyen una etapa de inversión extra, típicamente entre la etapa de entrada y el divisor de fase. Como resultado, las compuertas TTL no inversoras son por lo regular más grandes y más lentas que las compuertas inversoras en las que están basadas.

Como con CMOS, las compuertas TTL pueden ser diseñadas con salidas de tres estados. Tales compuertas tienen una entrada de "habilitación o deshabilitación de salida" que permite que la salida se coloque en un estado de alta impedancia donde ninguno de los transistores de salida se encuentra encendido.

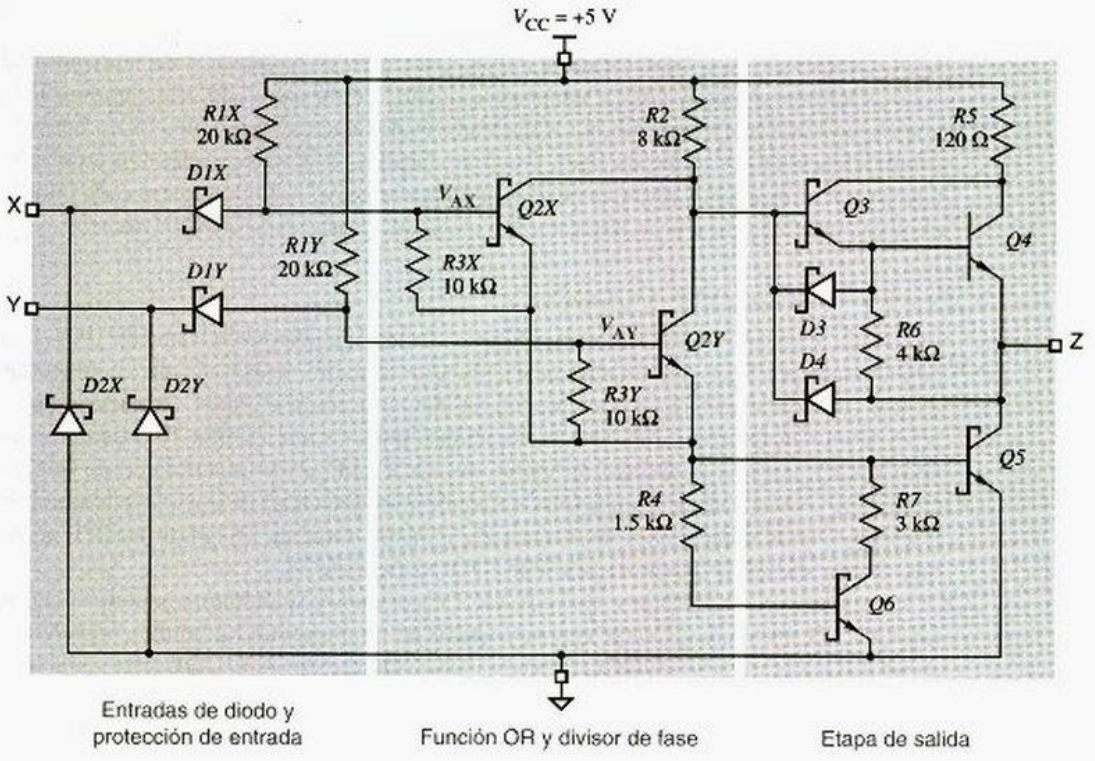


Figura 3-82 Diagrama de circuito de una compuerta NOR LS-TTL de dos entradas.

(a)

X	Y	V_{AX}	Q2X	V_{AY}	Q2Y	Q3	Q4	Q5	Q6	V_Z	Z
L	L	≤ 1.05	off	≤ 1.05	off	on	on	off	off	≥ 2.7	H
L	H	≤ 1.05	off	1.2	on	off	off	on	on	≤ 0.35	L
H	L	1.2	on	≤ 1.05	off	off	off	on	on	≤ 0.35	L
H	H	1.2	on	1.2	on	off	off	on	on	≤ 0.35	L

(b)

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Figura 3-83 Compuerta NOR LS-TTL de dos entradas: a) tabla de funciones; b) tabla de verdad; c) símbolo lógico.

salida de colector abierto

Algunas compuertas TTL se encuentran también disponibles con *salidas de colector abierto*. Tales compuertas omiten la mitad superior completa de la etapa de salida en la figura 3-75, de modo que solamente se proporciona arranque pasivo al estado ALTO mediante un resistor externo. Las aplicaciones y cálculos requeridos para las compuertas de compuerta abierta TTL son similares a las de las compuertas CMOS con salidas de drenaje abierto.

3.11 Familias TTL

Las familias TTL han evolucionado a través de los años en respuesta a las demandas de los diseñadores digitales por un mejor desempeño. Como resultado, tres familias TTL han llegado y se han marchado, y los diseñadores de la actualidad tienen cinco familias sobrevivientes de las cuales escoger. Todas las familias TTL son compatibles en el sentido de que utilizan el mismo voltaje de alimentación y niveles lógicos, pero cada familia tiene sus propias ventajas en términos de velocidad, consumo de energía y costos.

3.11.1 Familias TTL iniciales

La familia original TTL de compuertas lógicas fue introducida por Sylvania en 1963. Fue popularizada por Texas Instruments, cuyos números de parte "serie 7400" para compuertas y otros componentes TTL se convirtieron rápidamente en un estándar de la industria.

Como en la serie 7400 CMOS, los dispositivos en una familia TTL dada tienen números de parte de la forma 74FAM*nn*, donde "FAM" es un mnemónico de familia alfabético y *nn* es una denominación de función numérica. Los dispositivos en familias diferentes con el mismo valor de *nn* realizan la misma función. En la familia TTL original, "FAM" es nula y la familia se conoce como *TTL serie 74*.

TTL serie 74

Los valores de resistores en el circuito TTL original se cambiaron para obtener dos familias TTL más con diferentes características de rendimiento. La familia *74H (High-speed TTL, TTL de alta velocidad)* usaba valores de resistor más bajos para reducir el retardo de propagación a costa de un incremento en el consumo de energía. La familia *74L (Low-power TTL, TTL de baja potencia)* usaba valores de resistor más altos para disminuir el consumo de energía a costa del retardo de propagación.

74H (TTL de alta velocidad)

74L (TTL de baja potencia)

La disponibilidad de las tres familias TTL permitieron a los diseñadores digitales en la década de 1970, elegir entre alta velocidad y bajo consumo de energía para sus circuitos. Sin embargo, como mucha gente en esa década, ellos querían "¡tenerlo todo, y ya!" El desarrollo de los transistores Schottky proporcionó esta oportunidad e hizo obsoletos a los TTL 74, 74H y 74L. Las características de las familias TTL contemporáneas de mejor rendimiento se discuten en el resto de esta sección.

3.11.2 Familias TTL Schottky

Históricamente, la primera familia en hacer uso de los transistores Schottky fue la *74S (Schottky TTL)*. Con los transistores Schottky y valores de resistencia bajos, esta familia tenía mucho más alta velocidad, pero un consumo de energía más alto, que las series 74 TTL originales.

74S (TTL Schottky)

Quizás la familia TTL más ampliamente utilizada y ciertamente la menos costosa es la *74LS (Low-power Schottky TTL, Schottky TTL de baja potencia)*, introducida poco después de la 74S. Al combinar los transistores Schottky con valores de resistencia más altos, la 74LS TTL igualaba la velocidad de la serie 74 TTL pero tenía aproximadamente una quinta parte de su consumo de energía. Así, la 74LS se convirtió en una familia lógica preferida para nuevos diseños TTL.

74LS (TTL Schottky de baja potencia)

Las innovaciones subsecuentes de circuito y procesamiento de CI dieron surgimiento a otras dos familias lógicas Schottky. La familia *74AS (Advanced Schottky TTL)* ofrece velocidades aproximadamente dos veces más rápidas que la 74S casi con el mismo consumo de energía. La familia *74ALS (Advanced Low-power Schottky TTL)* ofrece tanto

74AS (TTL Schottky avanzado)

74ALS (TTL Schottky de baja potencia avanzado)

Tabla 3-11 Características de compuertas en familias TTL.

Descripción	Símbolo	Familia				
		74S	74LS	74AS	74ALS	74F
Retardo de propagación máximo (ns)		3	9	1.7	4	3
Consumo de energía por compuerta (mW)		19	2	8	1.2	4
Producto velocidad-energía (pJ)		57	18	13.6	4.8	12
Voltaje de entrada de nivel BAJO (V)	V_{ILmax}	0.8	0.8	0.8	0.8	0.8
Voltaje de salida de nivel BAJO (V)	V_{OLmax}	0.5	0.5	0.5	0.5	0.5
Voltaje de entrada de nivel ALTO (V)	V_{IHman}	2.0	2.0	2.0	2.0	2.0
Voltaje de salida de nivel ALTO (V)	V_{OHman}	2.7	2.7	2.7	2.7	2.7
Corriente de entrada de nivel BAJO (mA)	I_{ILmax}	−2.0	−0.4	−0.5	−0.2	−0.6
Corriente de salida de nivel BAJO (mA)	I_{OLmax}	20	8	20	8	20
Corriente de entrada de nivel ALTO (μA)	I_{IHmax}	50	20	20	20	20
Corriente de salida de nivel ALTO (μA)	I_{OHmax}	−1000	−400	−2000	−400	−1000

baja potencia como mayores velocidades que la 74LS y rivaliza con la 74LS en popularidad para requerimientos de propósito general en los nuevos diseños TTL. La familia *74F (Fast TTL)* se encuentra posicionada entre la 74AS y la 74ALS en el intercambio velocidad/potencia, y es probablemente la elección más popular para requerimientos de alta velocidad en los nuevos diseños TTL.

74F (TTL rápido)

3.11.3 Características de las familias TTL

Las características importantes de las familias TTL contemporáneas se resumen en la tabla 3-11. Los primeros dos renglones de la tabla enumeran el retardo de propagación (en nanosegundos) y el consumo de energía (en miliwatts) de una compuerta NAND de 2 entradas típica en cada familia.

La cifra de mérito de una familia lógica es su *producto velocidad-energía* expresado en el tercer renglón de la tabla. Como se discutió con anterioridad, éste es simplemente el producto del retardo de propagación por el consumo de energía de una compuerta típica. El producto velocidad-energía mide una clase de eficiencia (cuánta energía utiliza una compuerta lógica para conmutar su salida).

Los renglones restantes en la tabla 3-11 describen los parámetros de entrada y salida de las compuertas TTL típicas en cada una de las familias. Haciendo uso de esta información,

se puede analizar el comportamiento externo de las compuertas TTL sin conocer los detalles del diseño de circuito TTL interno. Estos parámetros fueron definidos y se discutieron en las secciones 3.10.2 y 3.10.3. Como de costumbre, las características de entrada y salida de componentes específicos pueden variar de los valores representativos dados en la tabla 3-11, de modo que usted siempre debe consultar el libro de datos del fabricante cuando analice un diseño real.

3.11.4 Una hoja de datos TTL

La tabla 3-12 muestra la parte de una típica hoja de datos del fabricante para el 74LS00. El 54LS00 mencionado en la hoja de datos es idéntico al 74LS00, excepto que tiene especificaciones para funcionamiento sobre el intervalo completo de temperatura y voltaje para aplicaciones militares, y su costo es mayor. La mayoría de las partes TTL tienen sus correspondientes versiones en serie 54 (militares). Se muestran tres secciones de la hoja de datos en la tabla:

condiciones recomendadas de operación

- *Condiciones de operación recomendadas* especifican el voltaje de alimentación, intervalos de voltaje de entrada, carga de salida de CD y valores de temperatura bajo los cuales el dispositivo funciona normalmente.

características eléctricas

- *Características eléctricas*, especifican voltajes y corrientes de CD adicionales que se observan en las entradas y salidas del dispositivo cuando funciona bajo las condiciones recomendadas:

  I_I Máxima corriente de entrada para un voltaje de entrada ALTO muy intenso.

  I_{OS} Corriente de salida con la salida ALTO en corto a tierra.

  I_{CCH} Corriente de alimentación cuando todas las salidas (en cuatro compuertas NAND) están en ALTO. (El número dado es para el encapsulado completo, que contiene cuatro compuertas NAND, de modo que la corriente por compuerta es una cuarta parte de la cantidad especificada).

  I_{CCL} Corriente de alimentación cuando todas las salidas (en cuatro compuertas NAND) están en BAJO.

características de conmutación

- *Características de conmutación* proporcionan retardos de propagación típicos y máximos bajo condiciones de operación "típicas" de $V_{CC} = 5$ V y $T_A = 25°C$. Un diseñador conservador debe incrementar estos retardos en un 5% a 10% para dar cuenta de diferentes temperaturas y voltajes de alimentación, e incluso más bajo condiciones pesadas de carga.

También se incluye una cuarta sección en el libro de datos del fabricante:

índices máximos absolutos

- *Índices absolutos máximos* indican las condiciones en el peor de los casos para la operación o almacenamiento del dispositivo, sin sufrir daños.

Un libro de datos completo también exhibe circuitos de prueba que son empleados para medir los parámetros cuando el dispositivo se fabrica, y gráficas que ilustran cómo los parámetros típicos varían de acuerdo con las condiciones de operación tales como el voltaje de alimentación (V_{CC}), temperatura ambiente (T_A) y carga (R_L, C_L).

Tabla 3-12 Hoja de datos típica del fabricante para el 74LS00.

CONDICIONES RECOMENDADAS DE OPERACIÓN								
		SN54LS00			SN74LS00			
Parámetro	**Descripción**	**Mínimo**	**Nominal**	**Máximo**	**Mínimo**	**Nominal**	**Máximo**	**Unidades**
V_{CC}	Voltaje de alimentación	4.5	5.0	5.5	4.75	5.0	5.25	V
V_{IH}	Voltaje de entrada de nivel ALTO	2.0			2.0			V
V_{IL}	Voltaje de entrada de nivel BAJO			0.7			0.8	V
I_{OH}	Corriente de salida de nivel ALTO			−0.4			−0.4	mA
I_{OL}	Corriente de salida de nivel BAJO			4			8	mA
T_A	Temperatura ambiente de operación	−55		125	0		70	°C

CARACTERÍSTICAS ELÉCTRICAS SOBRE EL INTERVALO DE TEMPERATURA AMBIENTE RECOMENDADO								
		SN54LS00			SN74LS00			
Parámetro	**Condiciones de prueba[1]**	**Mínimo**	**Típico[2]**	**Máximo**	**Mínimo**	**Típico[2]**	**Máximo**	**Unidades**
V_{IK}	V_{CC} = Min., I_N = −18 mA			−1.5			−1.5	V
V_{OH}	V_{CC} = Min., V_{IL} = Max., I_{OH} = −0.4 mA	2.5	3.4		2.7	3.4		V
V_{OL}	V_{CC} = Min., V_{IH} = 2.0 V, I_{OL} = 4 mA		0.25	0.4		0.25	0.4	V
	V_{CC} = Min., V_{IH} = 2.0 V, I_{OL} = 8 mA					0.35	0.5	
I_I	V_{CC} = Max., V_I = 7.0 V			0.1			0.1	mA
I_{IH}	V_{CC} = Max. V_I = 2.7 V			20			20	µA
I_{IL}	V_{CC} = Max. V_I = 0.4 V			−0.4			−0.4	mA
I_{OS}[3]	V_{CC} = Max.	−20		−100	−20		−100	mA
I_{CCH}	V_{CC} = Max., V_I = 0 V		0.8	1.6		0.8	1.6	mA
I_{CCL}	V_{CC} = Max., V_I = 4.5 V		2.4	4.4		2.4	4.4	mA

CARACTERÍSTICAS DE CONMUTACIÓN, V_{CC} = 5.0 V, T_A = 25°C						
Parámetro	**Desde (Entrada)**	**Hasta (Salida)**	**Condiciones de prueba**	**Mínimo**	**Típico**	**Máximo**	**Unidades**
t_{PLH}	A o B	Y	R_L = 2 kΩ, C_L = 15 pF		9	15	ns
t_{PHL}					10	15	

NOTAS:

1. *Para condiciones mostradas como Máximo o Mínimo, utilice el valor apropiado especificado bajo las Condiciones Recomendadas de Operación.*
2. *Todos los valores típicos están a V_{CC} = 5.0 V, T_A = 25°C.*
3. *No más de una salida debería ponerse en corto a la vez; la duración del cortocircuito no debería exceder un segundo.*

*3.12 Interfaz CMOS/TTL

Un diseñador digital selecciona una familia lógica "predeterminada" para utilizar en un sistema, basándose en requerimientos generales de velocidad, potencia, costo, y así sucesivamente. Sin embargo, el diseñador puede seleccionar dispositivos de otras familias en algunos casos debido a la disponibilidad u otros requerimientos especiales. (Por ejemplo, no todos los números de parte 74LS se encuentran disponibles en 74HCT, y viceversa.) De este modo, es importante para un diseñador entender las implicaciones de conectar salidas TTL a entradas CMOS, y al contrario.

Existen varios factores por considerar en las interfaces TTL/CMOS, y el primero es el margen de ruido. El margen de ruido de CD de estado BAJO depende del V_{OLmax} de la salida de control y del V_{ILmax} de la entrada controlada, y es igual a $V_{ILmax} - V_{OLmax}$. De manera semejante, el margen de ruido de CD de estado ALTO es igual a $V_{OHmin} - V_{IHmin}$. La figura 3-84 muestra los números relevantes para las familias TTL y CMOS.

Por ejemplo, el margen de ruido de CD del estado BAJO de HC o HCT controlando TTL es 0.8 – 0.33 = 0.47 V, y del estado ALTO es 3.84 – 2.0 = 1.84 V. Por otro lado, el margen de estado ALTO de TTL controlando HC o VHC es 2.7 – 3.85 = –1.15 V. En otras palabras, TTL controlando HC o CA no funciona, a menos que la salida ALTO de TTL sea mayor y que el umbral de entrada ALTO de CMOS sea menor por un total de 1.15 V, en comparación con sus especificaciones, en el peor de los casos. Para controlar las entradas CMOS apropiadamente a partir de salidas TTL, los dispositivos CMOS deberían ser HCT, VHCT o FCT en lugar de HC o VHC.

El siguiente factor a considerar es el fan out. Como con TTL puro (sección 3.10.3), un diseñador debe sumar los requerimientos de corriente de entrada de los dispositivos controlados por una salida y comparar con las capacidades de la salida en ambos estados. El fan out no es un problema cuando TTL controla a CMOS, puesto que las entradas CMOS casi no requieren corriente en ninguno de los estados. Por otra parte, las entradas TTL, especialmente en el estado BAJO, requieren una cantidad considerable de

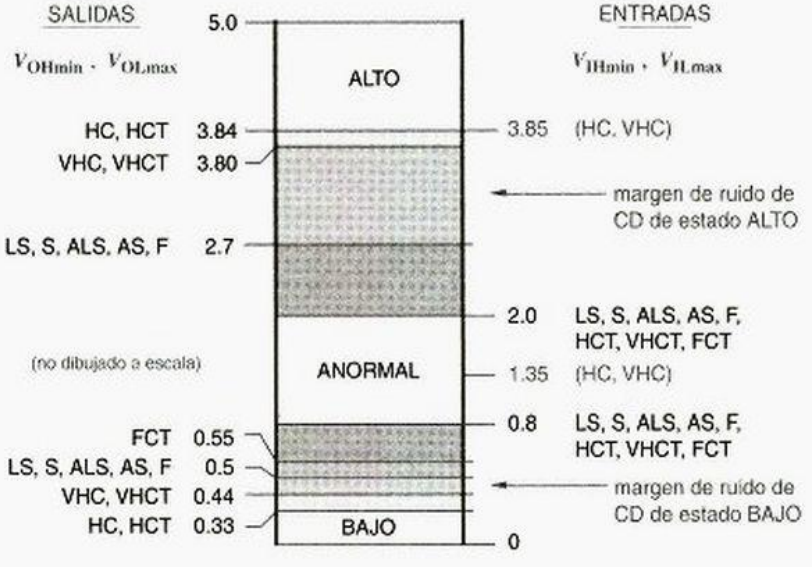


Figura 3-84
Niveles de salida y entrada para la interfaz de las familias TTL y CMOS. (Nótese que las entradas HC y VHC no son compatibles con TTL.)

corriente, especialmente en comparación con las capacidades de salida HC y HCT. Por ejemplo, una salida HC o HCT puede controlar 10 LS o solamente dos entradas S-TTL.

El último factor es la carga capacitiva. Hemos visto que la capacitancia de carga incrementa tanto el retardo como la disipación de energía de los circuitos lógicos. Los incrementos en el retardo son especialmente notables con salidas HC y HCT, cuyos tiempos de transición se incrementan en 1 ns por cada 5 pF de capacitancia de carga. Los transistores en salidas FCT tienen resistencias de "encendido" muy bajas, de modo que sus tiempos de transición aumentan solamente en alrededor de 0.1 ns por cada 5 pF de capacitancia de carga.

Para una capacitancia de carga dada, el voltaje de alimentación y aplicación de todas las familias CMOS tienen una disipación dinámica de energía semejante, puesto que cada variable en la ecuación CV^2f es la misma. Por otra parte, las salidas TTL tienen una disipación dinámica de energía ligeramente inferior, puesto que la excursión del voltaje entre los niveles TTL ALTO y BAJO es menor.

*3.13 Lógica CMOS de bajo voltaje e interfaz

Dos factores importantes han forzado a la industria para que utilice voltajes de alimentación más bajos en los dispositivos CMOS:

- En la mayoría de las aplicaciones, los voltajes de salida CMOS oscilan de línea a línea de manera que la variable V en la ecuación CV^2f es el voltaje de alimentación. La disminución del voltaje de alimentación reduce la disipación dinámica de energía.
- A medida que la industria tiende hacia geometrías de transistor aún más pequeñas, el aislamiento del óxido entre una compuerta de transistor CMOS y su fuente y drenaje se está haciendo cada vez más delgado, y por lo tanto se está haciendo incapaz de aislar potenciales de voltaje tan "altos" como 5 V.

Como resultado, JEDEC, un grupo de estándares de la industria de los circuitos integrados, seleccionó 3.3 V ± 0.3 V, 2.5 V ± 0.2 V y 1.8 V ± 0.15 V como los futuros voltajes de alimentación lógicos "estándar". Los estándares JEDEC especifican los niveles de voltaje lógico de entrada y salida para dispositivos que funcionan con estos voltajes de alimentación.

La emigración a voltajes menores se ha presentado en etapas y continuará haciéndolo así. Para familias lógicas discretas, la tendencia ha sido producir partes que funcionen y produzcan salidas a menores voltajes pero que toleren también entradas con mayor voltaje. Este enfoque ha permitido que las familias CMOS de 3.3 V funcionen con familias CMOS y TTL de 5 V, como veremos en la siguiente sección.

Muchos ASIC y microprocesadores han seguido un enfoque parecido, pero con frecuencia se utiliza también otro enfoque. Estos dispositivos son lo suficientemente grandes como para que pueda tener sentido suministrarlos con dos voltajes de alimentación. Un voltaje bajo, tal como 2.5 V, se suministra para hacer funcionar las compuertas internas del chip, o *lógica del núcleo*. Se proporciona un voltaje mayor, como 3.3 V, para hacer funcionar los circuitos de entrada y salida externos, o *anillo de alimentación auxiliar*, para mantener la compatibilidad con los dispositivos de generaciones antiguas que hay en el sistema. Se emplean circuitos especiales que incluyen buffers para transformar con seguridad y rapidez los voltajes lógicos del anillo de alimentación auxiliar y la lógica del núcleo.

lógica de núcleo
anillo terminal o anillo de alimentación auxiliar

*3.13.1 Lógica LVCMOS y LVTTL de 3.3 V

Las relaciones entre niveles de señal para dispositivos TTL estándar y CMOS de bajo voltaje que funcionan a sus voltajes nominales de alimentación se ilustran adecuadamente en la figura 3-85, adaptada de una nota de aplicación de Texas Instruments. Los niveles de señal simétricos, originales para familias CMOS de 5 V puros tales como HC y VHC se muestran en (a). Las familias CMOS compatibles con TTL tales como HCT, VHCT y FCT desplazan los niveles de voltaje hacia abajo para tener compatibilidad con TTL como se muestra en (b).

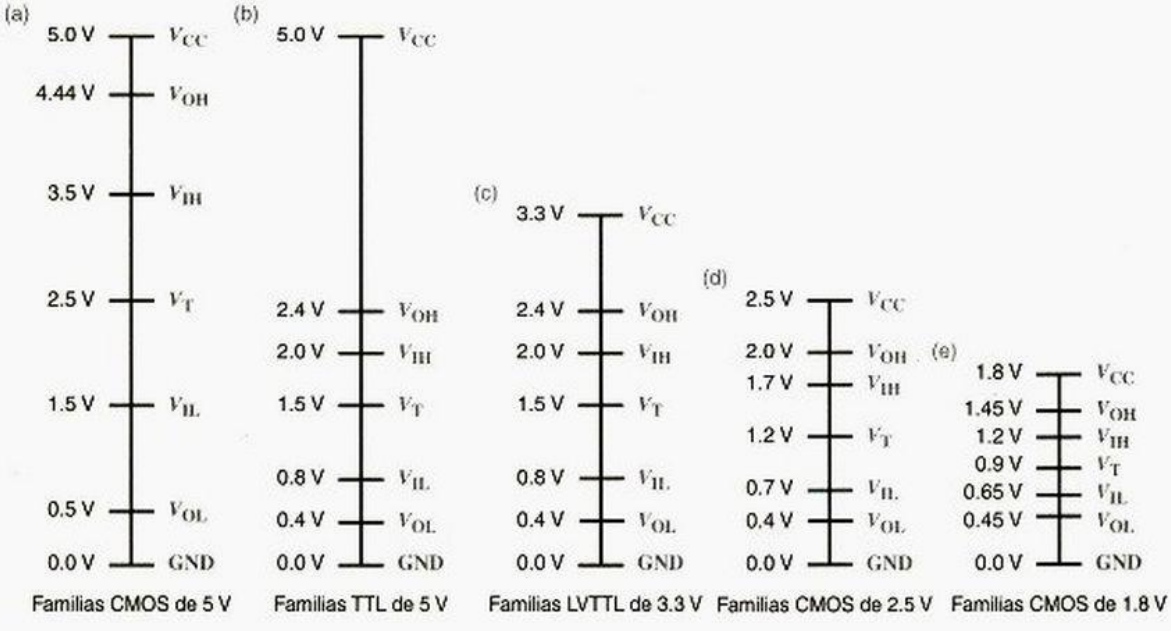
El primer paso en la progresión de voltajes de alimentación CMOS más bajos fue a 3.3 V. El estándar JEDEC para lógica de 3.3 V define en realidad dos conjuntos de niveles. Los niveles *LVCMOS (low voltaje CMOS, CMOS de bajo voltaje)* se utilizan en aplicaciones CMOS puras donde las salidas tienen cargas de CD ligeras (menores de 100 μA), de modo que V_{OL} y V_{OH} se mantienen dentro de 0.2 V de los rieles de alimentación. Los niveles *LVTTL (low voltaje TTL, TTL de bajo voltaje)*, mostrados en (c), se emplean en aplicaciones donde las salidas tienen cargas de CD significativas, de manera que V_{OL} puede ser tan alto como de 0.4 V y V_{OH} puede ser tan bajo como de 2.4 V.

LVCMOS (CMOS)

LVTTL (TTL)

El posicionamiento de los niveles lógicos TTL en el extremo inferior del intervalo de los 5 V fue realmente fortuito. Como se muestra en la figura 3-85(b) y (c), era posible definir los niveles LVTTL para que coincidieran exactamente con los niveles TTL. De este modo, una salida LVTTL puede controlar una entrada TTL sin ningún problema, mientras sus especificaciones de corriente de salida (I_{OLmax}, I_{OHmax}) sean respetadas. De manera similar, una salida TTL puede controlar una entrada LVTTL, excepto por el problema de controlarlo más allá del V_{CC} de 3.3 V del LVTTL, como se discute a continuación.

Figura 3-85 Comparación entre niveles lógicos: (a) CMOS de 5 V; (b) TTL de 5 V, Incluyendo 5-V TTL-compatible CMOS; (c) 3.3-V LVTTL; (d) 2.5-V CMOS; (e) 1.8-V CMOS.



*3.13.2 Entradas tolerantes a 5 V

Las entradas de una compuerta no necesariamente tolerarán voltajes mayores de V_{CC}. Éste es un problema cuando se utilizan familias tanto de 3.3 V como de 5 V en un sistema. Por ejemplo, los dispositivos CMOS de 5 V con facilidad producen salidas de 4.9 V cuando se encuentran ligeramente cargados, y tanto los dispositivos CMOS como TTL producen en forma rutinaria salidas de 4.0 V, aun cuando estén moderadamente cargadas. Las entradas de dispositivos de 3.3 V pueden no gustar de estos altos voltajes.

El voltaje máximo V_{Imax} que puede ser tolerado por una entrada se describe en la sección de "índices máximos absolutos" de la hoja de datos del fabricante. Para dispositivos HC, V_{Imax} es igual a V_{CC}. De este modo, si un dispositivo HC es energizado por una alimentación de 3.3 V, no puede ser controlado por ninguna de las salidas CMOS o TTL de 5 V. Para dispositivos VHC, por otra parte, V_{Imax} es de 7 V; así, los dispositivos VHC con una alimentación de 3.3 V pueden ser utilizados para convertir salidas 5 V a niveles de 3.3 V para su uso con microprocesadores de 3.3 V, memorias y otros dispositivos en un subsistema de 3.3 V puro.

La figura 3-86 explica por qué algunas entradas son tolerantes a 5 V y otras no lo son. Como se muestra en (a), la estructura de entrada HC y HCT en realidad contiene dos *diodos de fijación* inversamente polarizados, los cuales no habíamos mostrado antes, entre cada señal de entrada y V_{CC} y tierra. El propósito de estos diodos es derivar específicamente cualquier valor de señal de entrada transitoria menor que 0 a través de *D1*, o mayor que V_{CC} a través de *D2* para la correspondiente línea de alimentación. Tales transitorios pueden presentarse como un resultado de reflejos de línea de transmisión, como se describe en la sección 11.4. La derivación de "transitorio de bajo voltaje" o "sobreimpulso" a tierra o V_{CC} reduce la magnitud y duración de los reflejos.

diodos de fijación

Naturalmente, el diodo *D2* no puede distinguir entre un sobreimpulso transitorio y un voltaje de entrada persistente mayor que V_{CC}. De aquí que, si una salida de 5 V es conectada a una de estas entradas, no verá la muy alta impedancia normalmente asociada con una entrada CMOS. En su lugar, verá una trayectoria con relativamente baja impedancia hacia V_{CC} a través del diodo *D2* ahora directamente polarizado, y fluirá corriente excesiva.

La figura 3-86(b) muestra una entrada CMOS tolerante a 5 V. Esta estructura de entrada simplemente omite *D2*; el diodo *D1* todavía está proporcionado para fijar el sobreimpulso. Las familias VHC y AHC utilizan esta estructura de entrada.

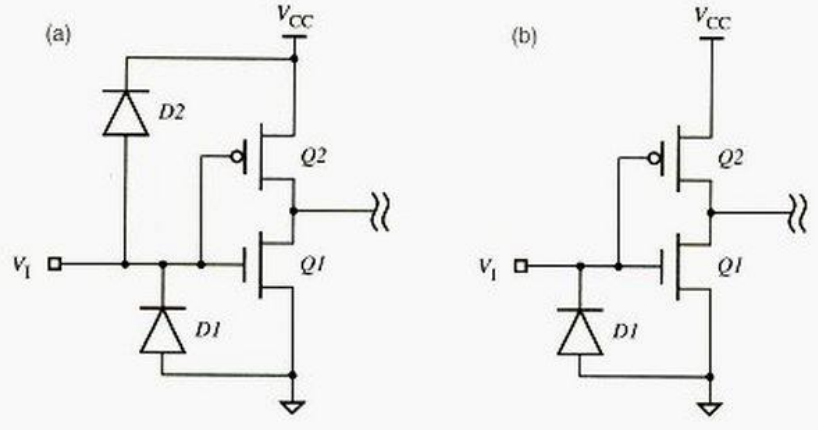


Figura 3-86
Estructuras de entrada CMOS: a) HC no tolerante a 5 V; b) VHC tolerante a 5 V.

La clase de estructura de entrada mostrada en la figura 3-86(b) es necesaria, pero no suficiente para crear entradas tolerantes de 5 V. Los transistores en el proceso de fabricación particular de un dispositivo también deben ser capaces de resistir potenciales de voltaje mayores que V_{CC}. Con esta base, V_{Imax} en la familia VHC se encuentra limitado a 7.0 V. En muchos procesos ASIC de 3.3 V, no es posible obtener entradas tolerantes a 5 V, incluso si usted está dispuesto a renunciar a los beneficios de línea de transmisión del diodo *D2*.

*3.13.3 Salidas tolerantes a 5 V

La tolerancia a 5 volts también debe ser considerada para las salidas, en particular cuando las salidas de 3.3 V y las de 5 V de tres estados se conectan a un bus. Cuando la salida de 3.3 V se encuentra en el estado deshabilitado Hi-Z, un dispositivo de 5 V puede estar controlando el bus, y una señal de 5 V puede aparecer en la *salida* del dispositivo de 3.3 V.

En esta situación, la figura 3-87 explica por qué algunas salidas son tolerantes a 5 V y otras no lo son. Como se ilustra en (a), la salida de tres estados CMOS estándar tiene un transistor *Q1* de canal-*n* hacia tierra y un transistor de canal-*p* *Q2* hacia V_{CC}. Cuando la salida es deshabilitada, la circuitería (que no se muestra) mantiene la compuerta de *Q1* cerca de los 0 V y la compuerta de *Q2* cerca de V_{CC}, de modo que ambos transistores se encuentran apagados y Y es Hi-Z.

Ahora considere lo que ocurre si V_{CC} es 3.3 V y un dispositivo diferente aplica una señal de 5 V a la terminal de salida Y en (a). Entonces el drenaje de *Q2*(Y) se encuentra a 5 V mientras que la compuerta (V_2) está todavía a solamente 3.3 V. Con la compuerta a un potencial más bajo que el drenaje, *Q2* comenzará a conducir y proporcionará una trayectoria de relativamente baja impedancia desde Y hacia V_{CC}, y habrá un flujo excesivo de corriente. Tanto las salidas de triple estado HC como VHC tienen esta estructura y por lo tanto no son tolerantes a 5 V.

La figura 3-87(b) muestra una estructura de salida tolerante a 5 V. Un transistor *Q3* de canal-*p* extra se utiliza para evitar que *Q2* se encienda cuando no deba hacerlo. Cuando V_{SAL} es mayor que V_{CC}, *Q3* se enciende. Esto forma una trayectoria de relativamente baja impedancia desde Y hasta la compuerta de *Q2*, el cual permanece ahora apagado porque su voltaje de compuerta V_2 ya no puede estar por abajo del voltaje de drenaje. Esta estructura de salida se emplea en la familia LVC (Low-Voltage CMOS, CMOS de bajo voltaje) de Texas Instruments.

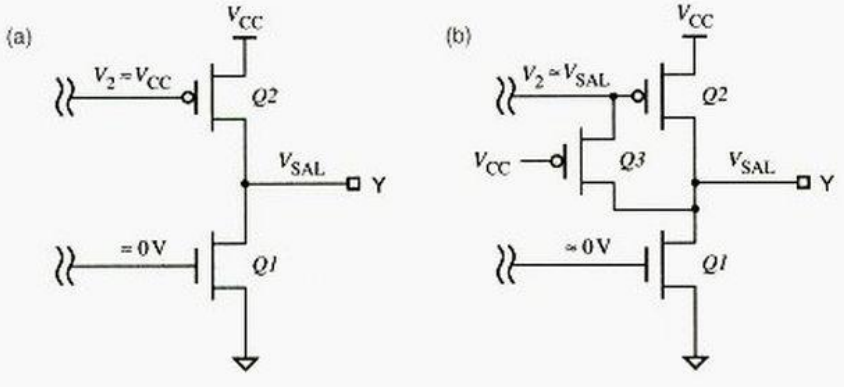


Figura 3-87
Estructuras de salida de triple estado CMOS: a) HC y VHC no tolerantes a 5 V; b) LVC tolerante a 5 V.

*3.13.4 Resumen de interfaces TTL/LVTTL

Basándonos en la información de las subsecciones anteriores, los dispositivos TTL (5 V) y LVTTL (3.3 V) se pueden mezclar en el mismo sistema sujetándose únicamente a tres reglas:

1. Las salidas LVTTL pueden controlar directamente entradas TTL, observando a las restricciones habituales de corriente de salida (I_{OLmax}, I_{OHmax}) de los dispositivos de control.
2. Las salidas TTL pueden controlar entradas LVTTL si las entradas son tolerantes a 5 V:
3. Las salidas de tres estados TTL y LVTTL pueden controlar el mismo bus si las salidas LVTTL son tolerantes a 5 V.

*3.13.5 Lógica de 2.5 V y de 1.8 V

La transición de la lógica de 3.3 V a la de 2.5 V no será tan fácil. Es verdad que las salidas de 3.3 V controlan entradas de 2.5 V mientras que las entradas sean tolerantes a 3.3 V. Sin embargo, un rápido examen a la figura 3-85(c) y (d) de la página 172 muestra que V_{OH} de una salida de 2.5 V es igual a V_{IH} de una entrada de 3.3 V. En otras palabras, existe un margen de ruido de CD de estado ALTO cero cuando una salida de 2.5 V controla una entrada de 3.3 V; una situación nada buena.

traductor de nivel
desplazador de nivel

La solución a este problema es utilizar un *traductor de nivel* o *desplazador de nivel*, un dispositivo que está energizado por ambos voltajes de alimentación y que internamente acelera los niveles lógicos más bajos (2.5 V) hasta los más altos (3.3 V). Muchos de los ASICs y microprocesadores de la actualidad contienen traductores de nivel internamente, permitiéndoles funcionar con un núcleo de 2.5 V o 2.7 V y un anillo terminal de 3.3 V, como discutimos al principio de esta sección. Si esperamos a que los dispositivos discretos de 2.5 V lleguen a ser populares, podremos esperar que los principales fabricantes de semiconductores produzcan traductores de nivel así como componentes por separado.

El siguiente paso será una transición de la lógica de 2.5 V a la de 1.8 V. Haciendo referencia a la figura 3-85(d) y (e), se puede ver que el margen de ruido de CD para el estado ALTO es en realidad negativo cuando una salida de 1.8 V maneja una entrada de 2.5 V, por lo tanto en este caso también se necesitan traductores de nivel.

*3.14 Lógica de emisor acoplado

La clave para reducir el tiempo de retardo de propagación en una familia lógica bipolar es evitar que los transistores de una compuerta se saturen. En la sección 3.9.5, aprendimos la forma en la que los diodos Schottky evitan la saturación en compuertas TTL.Sin embargo, también se puede evitar la saturación mediante el uso de una estructura de circuito radicalmente diferente, denominada *lógica de modo de corriente (CML, current-mode logic)* o *lógica de emisor acoplado (ECL, emitter-coupled logic)*.

lógica de modo de corriente (CML)

lógica de emisor acoplado (ECL)

A diferencia de las otras familias lógicas que se presentan en este capítulo, la lógica CML no produce una gran oscilación de voltaje entre los niveles BAJO y ALTO. En vez de ello, tiene una pequeña oscilación de voltaje menor a un volt, y conmuta internamente la corriente entre dos trayectorias posibles, dependiendo del estado de la salida.

La primera familia lógica CML fue introducida por General Electric en 1961. El concepto pronto fue refinado por Motorola y otros para producir las todavía populares familias de *lógica de emisor acoplado (ECL, emitter-coupled logic)* de 10K y 100K. Estas

lógica de emisor acoplado (ECL)

familias son extremadamente rápidas, ofrecen retardos de propagación tan breves como 1 ns. La más reciente familia ECL, ECLinPS (literalmente, ECL en picosegundos), ofrece retardos máximos inferiores a 0.5 ns (500 ps), e incluye las funciones de activación y desactivación del retardo de señal en el mismo encapsulado del CI. A lo largo de la evolución de la tecnología de los circuitos digitales, algún tipo de ECL siempre ha sido la tecnología más rápida para componentes lógicos que tienen encapsulados discretos.

A pesar de ello, las familias comerciales ECL no son ni con mucho tan populares como CMOS y TTL, principalmente a consecuencia de que consumen una mayor cantidad de energía. De hecho, el gran consumo de energía hace del diseño de supercomputadoras ECL, tal como la Cray-1 y la Cray-2, constituye todo un reto tanto en tecnología de enfriamiento como en diseño digital. Asimismo, ECL tiene un producto velocidad-energía deficiente, no proporciona un alto nivel de integración, tiene velocidades de borde rápidas que requieren diseño para efectos de línea de transmisión en la mayoría de las aplicaciones y no es directamente compatible con las lógicas TTL y CMOS. No obstante, la lógica ECL aún mantiene su sitio como una tecnología lógica y de interfaz en muchos equipos de comunicaciones de alta velocidad, incluyendo las interfaces transceptoras de fibra óptica para redes Ethernet de gigabits y de modo de transferencia asincrónica (ATM, Asynchronous Transfer Mode)

*3.14.1 Circuito básico CML

La idea básica de la lógica de modo de corriente se ilustra en el circuito del inversor/buffer de la figura 3-88. Este circuito tiene tanto una salida inversora (SAL1) como una salida no inversora (SAL2). Dos transistores se encuentran conectados como un *amplificador diferencial* con un resistor de emisor común. Los voltajes de alimentación para este ejemplo son $V_{CC} = 5.0$, $V_{BB} = 4.0$ y $V_{EE} = 0$ V, y los niveles de entrada BAJO y ALTO se definen como 3.6 y 4.4 V. Este circuito en realidad produce niveles de salida BAJO y ALTO que son mayores por 0.6 V (4.2 y 5.0 V), pero esto se corrige en los circuitos ECL reales.

amplificador diferencial

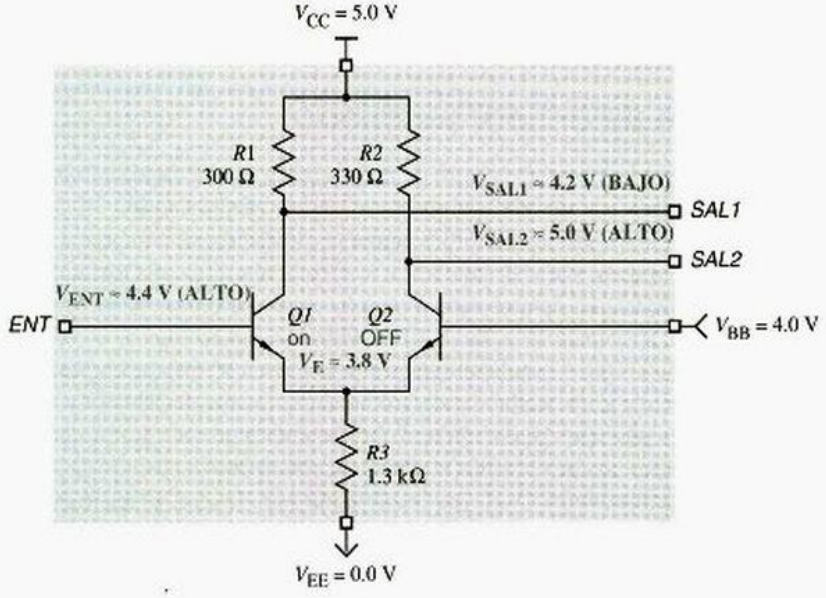


Figura 3-88
Circuito básico CML inverso/buffer con entrada en ALTO.

Cuando V_{ENT} se encuentra en ALTO, como se muestra en la figura, el transistor *Q1* está encendido, pero no saturado, y el transistor *Q2* está apagado. Esto es verdad gracias a una cuidadosa elección de los valores del resistor y niveles de voltaje. De este modo, V_{SAL2} se lleva hasta 5.0 V (ALTO) a través de *R2* y puede demostrarse que la caída de voltaje a través de *R1* es de aproximadamente 0.8 V, de manera que V_{SAL1} es cercana a 4.2 V (BAJO).

Cuando V_{ENT} está en BAJO, como se muestra en la figura 3-89, el transistor *Q2* está encendido, pero no saturado, y el transistor *Q1* está apagado. De este modo, V_{SAL1} se lleva a 5.0 V a través de *R1* y puede demostrarse que V_{SAL2} es aproximadamente 4.2 V.

salidas diferenciales

Las salidas de este inversor se conocen como *salidas diferenciales* porque son siempre complementarias, y es posible determinar el estado de salida al examinar la diferencia entre los voltajes de salida ($V_{SAL1} - V_{SAL2}$) en vez de sus valores absolutos. Es decir, la salida es 1 si ($V_{SAL1} - V_{SAL2}$) > 0, y es 0 si ($V_{SAL1} - V_{SAL2}$) < 0. Es posible construir circuitos de entrada con dos alambres por entrada lógica que definen el valor de señal lógica de esta manera; se conocen como *entradas diferenciales*.

entradas diferenciales

Se utilizan señales diferenciales en la mayoría de las aplicaciones de "interfaz" y "distribución de reloj" ECL debido a su bajo sesgo y alta inmunidad al ruido. Son de "bajo sesgo" porque la temporización de una transición de 0 a 1 o de 1 a 0 no depende de manera crítica de los umbrales de voltaje, los cuales pueden cambiar con la temperatura o entre dispositivos. En vez de ello, la temporización depende solamente de cuando los voltajes se cruzan de manera relativa entre sí. De modo semejante, la definición "relativa" de 0 y 1 proporciona una inmunidad excepcional al ruido, puesto que el ruido que generan las variaciones de voltaje en la fuente de alimentación o las señales de ruido que se acoplan desde fuentes externas tienden a ser una *señal en modo común* que afecta ambas señales diferenciales de modo semejante, dejando el valor de la diferencia sin cambio alguno.

señal en modo común

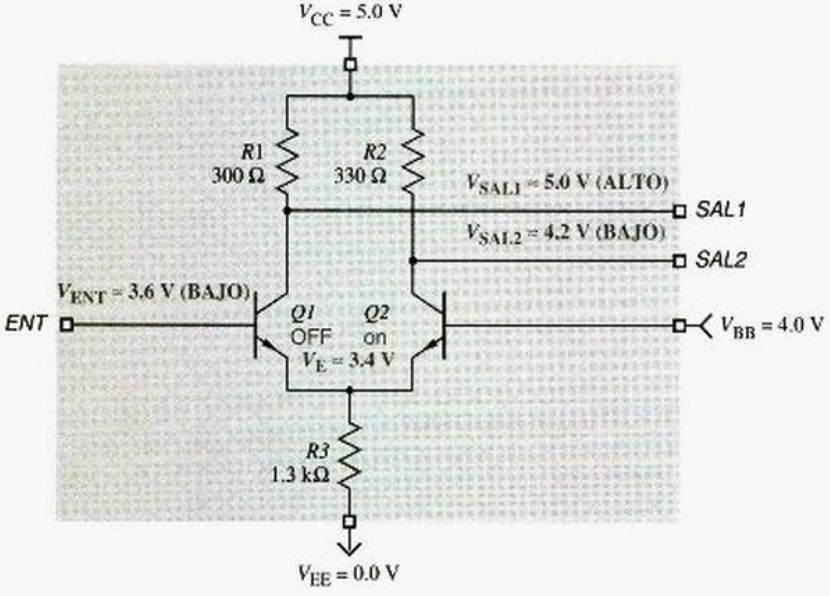


Figura 3-89
Circuito básico CML inversor/buffer con entrada en ALTO.

entrada asimétrica

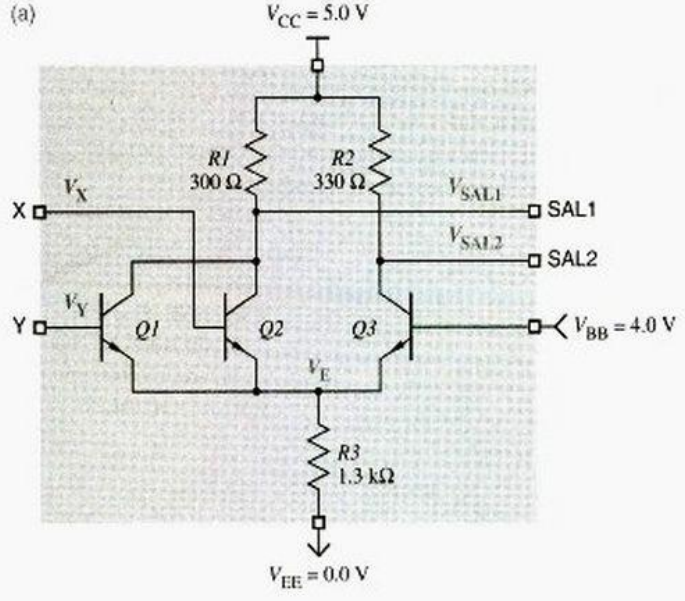
También es posible, naturalmente, determinar el valor lógico al detectar el nivel de voltaje absoluto de una señal de entrada, denominada *entrada asimétrica*. Las señales asimétricas se utilizan en la mayor parte de las aplicaciones "lógicas" ECL para evitar el gasto que implica duplicar la cantidad de líneas de señal. El inversor CML básico en la figura 3-89 tiene una entrada asimétrica. Siempre tiene ambas "salidas" disponibles en forma interna; el circuito es en realidad un buffer inversor o no inversor, dependiendo de si utilizamos SAL1 o bien SAL2.

Para realizar funciones lógicas con el circuito básico de la figura 3-89, simplemente colocamos transistores adicionales en paralelo con *Q1*, de modo semejante al enfoque en una compuerta NOR TTL. Por ejemplo, la figura 3-90 muestra una compuerta OR/NOR CML de 2 entradas. Si cualquiera de las entradas está en ALTO, el transistor de entrada correspondiente se encuentra activo, y V_{SAL1} está en BAJO (salida NOR). Al mismo tiempo, *Q3* se encuentra apagado y V_{SAL2} está en ALTO (salida OR).

Recuerde que los niveles de entrada para el inversor/buffer están definidos como de 3.6 y 4.4 V, mientras que los niveles de salida que producen son 4.2 y 5.0 V. Esto

Figura 3-90 Compuerta OR/NOR CML de 2 entradas: a) diagrama de circuito; b) tabla de funciones; c) símbolo lógico; d) tabla de verdad.

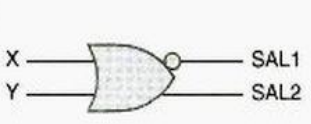
(a)



(b)

X	Y	V_X	V_Y	*Q1*	*Q2*	*Q3*	V_E	V_{SAL1}	V_{SAL2}	SAL1	SAL2
L	L	3.6	3.6	off	off	on	3.4	5.0	4.2	H	L
L	H	3.6	4.4	off	on	off	3.8	4.2	5.0	L	H
H	L	4.4	3.6	on	off	off	3.8	4.2	5.0	L	H
H	H	4.4	4.4	on	on	off	3.8	4.2	5.0	L	H

(c)



(d)

X	Y	SAL1	SAL2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

es obviamente un problema. Podríamos poner un diodo en serie con cada salida para disminuirla en 0.6 V para satisfacer los niveles de entrada, pero todavía nos deja otro problema: las salidas tienen un fan out deficiente. Una salida ALTO debe suministrar corriente de base a las entradas que controla, y esta corriente produce una caída de voltaje adicional a través de *R1* o *R2*, reduciendo el voltaje de salida (y no tenemos mucho margen para trabajar con él). Estos problemas se resuelven en las familias comerciales ECL, tal como la familia 10K que se describe a continuación.

*3.14.2 Familias ECL 10K/10H

Los componentes encapsulados en la familia ECL más popular de la actualidad tienen números de parte que están integrados por 5 dígitos, con la forma "10xxx" (por ejemplo, 10102, 10181, 10209), de modo que la familia se denomina en forma genérica *ECL 10K*. Esta familia tiene diversas mejoras sobre el circuito CML básico que describimos con anterioridad:

familia ECL 10K

- Una etapa de salida de seguidor de emisor que desplaza los niveles de salida para satisfacer los niveles de entrada y proporcionar una capacidad de control de corriente muy elevada, hasta de 50 mA por salida. Esto es también responsable del nombre dado a la familia lógica de "emisor acoplado".
- Una red de polarización interna que proporciona el voltaje V_{BB} sin necesidad de una fuente de alimentación externa independiente.
- La familia está diseñada para funcionar con $V_{CC} = 0$ (tierra) y $V_{EE} = -5.2$ V. En la mayoría de las aplicaciones, las señales de tierra están más libres de ruido que las señales de alimentación. En ECL, las señales lógicas están referenciadas al riel de mayor voltaje de alimentación algebraicamente, de modo que los diseñadores de la familia optaron por utilizar 0 V (una tierra "limpia") y emplear un voltaje negativo para V_{EE}. El ruido de alimentación que aparece en V_{EE} es una señal de "modo común" que es rechazada por el amplificador diferencial de la estructura de entrada.
- Las partes con el prefijo 10H (la *familia ECL 10H*) están completamente compensadas en voltaje, de modo que trabajarán adecuadamente con otros voltajes de alimentación aparte de $V_{EE} = -5.2$ V, como discutiremos en la sección 3.14.4.

familia ECL 10H

Los niveles lógicos BAJO y ALTO están definidos en la familia ECL 10K como se muestra en la figura 3-91. Nótese que aun cuando la alimentación es negativa, ECL asigna los nombres BAJO y ALTO a los voltajes que son *algebraicamente* más bajos y más altos, respectivamente.

Los márgenes de ruido de CD en ECL 10K son mucho menores que en CMOS y TTL, sólo 0.155 V en el estado BAJO y 0.125 V en el estado ALTO. Sin embargo, las compuertas ECL no requieren tanto margen de ruido como estas familias. A diferencia de CMOS y TTL, una compuerta ECL genera muy poco ruido de alimentación y de tierra cuando cambia de estado; su requerimiento de corriente permanece constante a medida que dirige la corriente de una trayectoria a otra. Además, las salidas tipo seguidor de emisor de la lógica ECL tienen muy poca impedancia en cualquier estado, y es difícil acoplar ruido desde una fuente externa a una línea de señal que está controlada por una salida de baja impedancia de esta clase.

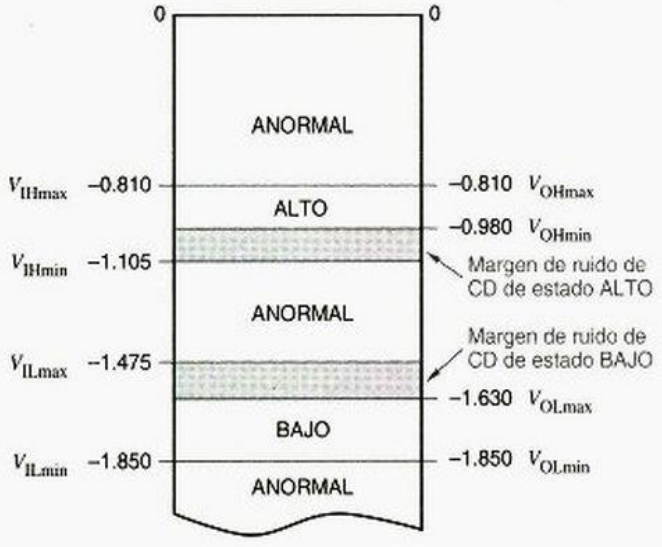


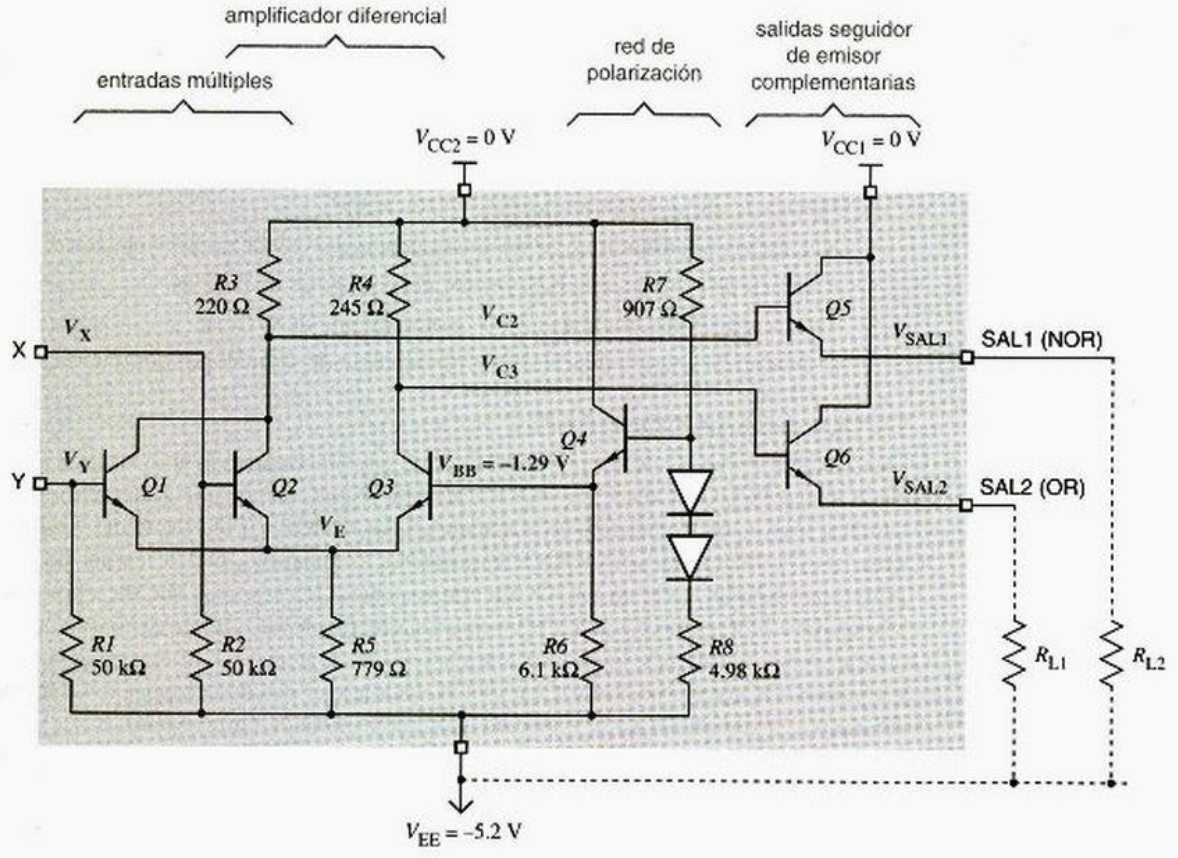
Figura 3-91
Niveles lógicos ECL 10 K.

La figura 3-92(a) muestra el circuito de una compuerta OR/NOR ECL, una sección de una compuerta cuádruple OR/NOR que tiene el número de parte 10102. Existe un resistor de disminución (pull-down) en cada entrada, el que permite tratar a cada entrada que no está conectada como un nivel BAJO. Los valores de los componentes de la red de polarización se selecionaron para generar $V_{BB} = -1.29$ V, lo que permite el funcionamiento adecuado del amplificador diferencial. Cada transistor de salida, que utiliza la configuración seguidor de emisor, mantiene su voltaje de emisor igual a una caída de voltaje del diodo por debajo de su voltaje de base, logrando por ello el desplazamiento de nivel que requiere la salida. La figura 3-92(b) resume la operación eléctrica de la compuerta.

Las salidas en configuración seguidor de emisor que se utilizan en ECL 10K requieren de resistores externos de disminución (external pull-down resistors), como se muestra en la figura. La familia 10K está diseñada para emplear resistores externos de disminución en lugar de resistores internos por buenas razones. Los tiempos de ascenso y caída de las transiciones de salida ECL son tan rápidos (por lo regular de 2 ns) que cualquier conexión de más de unas cuantas pulgadas (de longitud) debe tratarse como si fuera una línea de transmisión y debe terminar como se analiza en la sección 11.4. En lugar de gastar energía con un resistor interno de disminución, los dispositivos ECL 10K permiten la selección de un resistor externo que cumpla tanto los requerimientos de disminución como de línea de transmisión. La terminación más simple, adecuada para conexiones breves, consiste en conectar un resistor cuyo valor está en el intervalo de 270 Ω a 2 kΩ desde cada salida hacia V_{EE}.

Una compuerta típica ECL 10K tiene un retardo de propagación de 2 ns, comparable al 74AS TTL. Cuando sus salidas no se conectan, una compuerta 10K consume alrededor de 26 mW de potencia, también comparable a una compuerta 74AS TTL, la cual consume aproximadamente 20 mW. Sin embargo, la terminación requerida por ECL 10K también consume energía, de 10 a 150 mW por salida dependiendo del tipo de circuito de terminación. Una salida 74AS TTL puede o no requerir un circuito de terminación que consume energía, dependiendo de las características físicas de la aplicación.

(a)



(b)

X	Y	V_X	V_Y	Q1	Q2	Q3	V_E	V_{C2}	V_{C3}	V_{SAL1}	V_{SAL2}	SAL1	SAL2
L	L	−1.8	−1.8	off	off	on	−1.9	−0.2	−1.2	−0.9	−1.8	H	L
L	H	−1.8	−0.9	off	on	off	−1.5	−1.2	−0.2	−1.8	−0.9	L	H
H	L	−0.9	−1.8	on	off	off	−1.5	−1.2	−0.2	−1.8	−0.9	L	H
H	H	−0.9	−0.9	on	on	off	−1.5	−1.2	−0.2	−1.8	−0.9	L	H

(c)

X	Y	SAL1	SAL2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

(d)

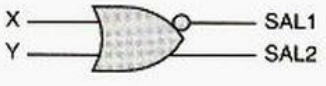


Figura 3-92 Compuerta OR/NOR ECL 10K de dos entradas: a) diagrama de circuito; b) tabla de funciones; c) tabla de verdad; d) símbolo lógico.

*3.14.3 Familia ECL 100K

familia ECL 100K

Los números de parte de los miembros de la familia *ECL 100K* están integrados por 6 dígitos y tienen la forma "100xxx" (por ejemplo, 100101, 100117, 100170), pero en general sus funciones son diferentes si se comparan con las funciones de las partes correspondientes 10K que tienen números similares. La familia 100K tiene las siguientes diferencias principales (con respecto a la familia 10K):

- Voltaje de alimentación reducido, $V_{EE} = -4.5$ V.
- Niveles lógicos diferentes, como consecuencia del voltaje de alimentación diferente.
- Retardos de propagación más cortos, por lo regular de 0.75 ns.
- Tiempos de transición más cortos, generalmente de 0.70 ns.
- Consumo de energía más elevado, típicamente de 40 mW por compuerta.

*3.14.4 ECL positivo (PECL)

Describimos la ventaja de la inmunidad al ruido que proporciona la alimentación negativa de la lógica ECL ($V_{EE} = -5.2$ o -4.5 V), pero también tenemos una gran desventaja: las familias lógicas más populares de la actualidad, los CMOS y TTL, los ASIC y microprocesadores, utilizan todos un voltaje de alimentación positivo, por lo regular de +5.0 V pero con tendencia a ser de +3.3 V. En consecuencia, los sistemas que incorporan tanto dispositivos ECL como CMOS/TTL requieren dos fuentes de alimentación (una positiva y otra negativa). Además, la interfaz entre los niveles lógicos ECL 10K o 100K negativos estándar y los niveles positivos CMOS/TTL requiere el uso de componentes especiales de traducción de nivel que se conecten a ambas fuentes de alimentación.

ECL positivo (PECL)

El *ECL positivo (PECL, Positive ECL, pronunciado como "pecl")* utiliza una fuente de alimentación estándar de +5.0 V. Nótese que no hay nada en el diseño de circuito ECL 10K de la figura 3-92 que requiera que V_{CC} se encuentre aterrizado, y que V_{EE} se conecte a una alimentación de –5.2 V. El circuito funcionará exactamente de la misma manera con V_{EE} conectado a tierra y V_{CC} a una alimentación de +5.2 V.

De este modo, los componentes PECL son simplemente componentes ECL estándar con V_{EE} conectado a tierra y V_{CC} a una fuente de alimentación de +5.0 V. El voltaje entre V_{EE} y V_{CC} es un poco menor que con el ECL 10K estándar y más que con el ECL 100K estándar, pero las partes de la serie 10H y 100K se encuentran compensadas en voltaje, y están diseñadas para trabajar bien aun cuando el voltaje de la fuente de alimentación es un poco alto o bajo.

Como los niveles lógicos de ECL, los niveles PECL estan referidos con respecto a V_{CC}, de modo que el nivel ALTO de PECL es de aproximadamente $V_{CC} - 0.9$ V y el BAJO es de alrededor de $V_{CC} - 1.7$ V, o aproximadamente 4.1 V y 3.3 V con un V_{CC} nominal de 5 V. Puesto que estos niveles están referenciados con respecto a V_{CC}, se mueven hacia arriba y hacia abajo con cualquier variación en V_{CC}. De esta forma, los diseños PECL requieren que se preste particular atención a cuestiones de distribución de energía, para evitar que el ruido sobre V_{CC} altere los niveles lógicos que se transmiten y reciben mediante dispositivos PECL.

Recuerde que los dispositivos CML/ECL producen salidas diferenciales y pueden tener entradas diferenciales. Una entrada diferencial es relativamente insensible a los niveles de voltaje absoluto de un par de señales de entrada, y sólo es sensible a su diferencia. Por consiguiente, las señales diferenciales pueden utilizarse con bastante eficacia en aplicaciones PECL para mejorar los niveles de ruido que se mencionaron en el párrafo anterior.

También es bastante común proporcionar entradas y salidas diferenciales compatibles con la lógica PECL en los dispositivos CMOS, lo que permite una interfaz directa entre el dispositivo CMOS y algún otro dispositivo como un transceptor de fibra óptica que utiliza niveles ECL o PECL. De hecho, a medida que los circuitos CMOS han emigrado a las fuentes de alimentación de 3.3 V, incluso ha sido posible construir entradas y salidas diferenciales tipo PECL que están referenciadas simplemente a la alimentación de 3.3 V, en lugar de hacerlo con una alimentación de 5 V.

Referencias

Los estudiantes que necesiten estudiar los fundamentos tal vez desean querer consultar "Electrical Circuits Review" de Bruce M. Fleischer. Este tutorial de 20 páginas abarca todos los conceptos fundamentales de un circuito que se utilizaron en este capítulo que también aparece como apéndice en la primera edición de la obra y como un archivo tipo " .pdf" en la página Web de este libro, `www.ddpp.com`.

Después de ver los resultados de la sorprendente marcha de las últimas décadas en el desarrollo de la electrónica digital, es fácil olvidar que los circuitos lógicos tenían un importante sitio en tecnologías que llegaron antes del transistor. En el capítulo 5 de *Introduction to the Methodology of Switching Circuits* (Van Nostrand, 1972), George J. Klir nos muestra cómo la lógica puede ser (y ha sido) realizada por una variedad de dispositivos físicos, incluyendo relevadores, tubos de vacío y sistemas neumáticos.

Si usted está interesado en más historia, una bella introducción a todo lo referente a las primeras familias lógicas bipolares puede encontrarse en *Logic Design with Integrated Circuits* por William E. Wickes (Wiley-Interscience, 1968). La introducción clásica a las características eléctricas TTL aparecieron en *The TTL Applications Handbook*, editado por Peter Alfke e Ib Larsen (Fairchild Semiconductor, 1973). Los primeros diseñadores lógicos también disfrutarán de *The TTL Cookbook* por Don Lancaster.

Para otra perspectiva acerca del material electrónico en este capítulo, se puede consultar casi cualquier texto de electrónica moderna. Muchos contienen una discusión mucho más analítica del funcionamiento de los circuitos digitales; por ejemplo, véase *Microelectronics* de J. Millman y A. Grabel (McGraw-Hill, 1987, segunda edición). Otra buena introducción a los circuitos integrados y a importantes familias lógicas se halla en *VLSI System Design* de Saburo Muroga (Wiley, 1982). Para circuitos NMOS y CMOS en particular, dos buenos libros son *Introduction to VLSI Systems* de Carver Mead y Lynn Conway (Addison-Wesley, 1980) y *Principles of CMOS VLSI Design* de Neil H. E. Weste y Kamran Eshraghian (Addison-Wesley, 1993).

Una introducción desenfadada y muy disfrutable a los chips digitales se encuentra en la obra de Clive Maxfield *Bebop to the Bolean Boggie* (LLH Technology Publishing, `www.llh-publishing.com`.) ¡Algunas personas piensan que la receta de mariscos en el Apéndice H vale por sí sola el precio del libro! Aun sin la receta, el libro es un clásico bien ilustrado que lo guía a través de los fundamentos de la electrónica digital, sus componentes y procesos.

Un entendimiento a fondo de los aspectos eléctricos del funcionamiento de los circuitos digitales es obligatorio para tener éxito en el diseño de circuitos de alta velocidad. Sin duda alguna el mejor libro sobre este tema es *High-Speed Digital Design: A Handbook of Black Magic*, de Howard Johnson y Martin Graham (Prentice Hall, 1993). Combina

Tabla 3-13 Libros de datos lógicos de los fabricantes.

Fabricante	*Número de orden*	*Tópicos*	*Título*	*Año*
Texas Instruments	SDLD001	74, 74S, 74LS TTL	*TTL Logic Data Book*	1988
Texas Instruments	SDAD001C	74AS, 74ALS TTL	*ALS/AS Logic Data Book*	1995
Texas Instruments	SDFD001B	74F TTL	*F Logic Data Book*	1994
Texas Instruments	SCLD001D	74HC, 74HCT CMOS	*HC/HCT Logic Data Book*	1997
Texas Instruments	SCAD001D	74AC, 74ACT CMOS	*AC/ACT Logic Data Book*	1997
Texas Instruments	SCLD003A	74AHC, 74AHCT CMOS	*AHC/AHCT Logic Data Book*	1997
Motorola	DL121/D	74F, 74LSTTL	*Fast and LSTTL Data*	1989
Motorola	DL129/D	74HC, 74HCT	*High-Speed CMOS Data*	1988
Motorola	DL138/D	74AC, 74ACT	*FACT Data*	1988
Motorola	DL122/D	10K ECL	*MECL Device Data*	1989

sólidos principios en electrónica con una impresionante perspicacia y experiencia en el diseño de sistemas digitales prácticos.

Las características de las familias lógicas de la actualidad pueden encontrarse en los libros de datos publicados por los fabricantes de dispositivos. Tanto Texas Instruments como Motorola publican extensos libros de datos para dispositivos TTL y CMOS, como los que se numeran en la tabla 3-13. Ambos fabricantes mantienen versiones actualizadas de sus libros de datos en la Web, `www.ti.com` y `www.mot.com`. Motorola también proporciona una agradable introducción al diseño ECL en el *MECL System Design Handbook* (publicación HB205, rev. 1, 1988).

Los estándares JEDEC (Joint Electron Device Engineering Council) para los niveles de lógica digital se hallan en el sitio Web de JEDEC, `www.jedec.org`. Los estándares JEDC para las lógicas de 3.3 V, 2.5 V y 1.8 V fueron publicados en 1994, 1995 y 1997, respectivamente.

Problemas propuestos

3.1 Una familia lógica particular define una señal de nivel BAJO en el intervalo 0.0–0.8 V y una señal de nivel ALTO en el intervalo 2.0–3.3 V. Bajo una convención de lógica positiva, indique el valor lógico asociado con cada uno de los siguientes niveles de señal:

(a) 0.0 V (b) 3.0 V (c) 0.8 V (d) 1.9 V

(e) 2.0 V (f) 5.0 V (g) −0.7 V (h) −3.0 V

3.2 Repita el problema 3.1 utilizando una convención de lógica negativa.

3.3 Discuta cómo un buffer lógico es diferente de un amplificador de audio.

3.4 ¿Es un buffer equivalente a una compuerta AND de una entrada o a una compuerta OR de una entrada?

3.5 Verdadero o falso: para un conjunto dado de valores de entrada, una compuerta NAND produce la salida opuesta de una compuerta NOR.

3.6 Escriba dos definiciones completamente diferentes de "compuerta" que se hayan utilizando en este capítulo.

3.7 ¿Qué clase de transistores se utilizan en las compuertas CMOS?

3.8 (Sólo para aficionados). Dibuje un circuito equivalente para un inversor CMOS empleando un relevador de polo simple y doble disparo.

3.9 Para un área de silicio dada, ¿cuál es probablemente más rápida, una compuerta NAND CMOS o una NOR CMOS?

3.10 Defina "fan in" y "fan out". ¿Cuál es probable que tenga que calcular?

3.11 Dibuje el diagrama de circuito, la tabla de funciones y el símbolo lógico para una compuerta NOR CMOS de tres entradas en el estilo de la figura 3-16.

3.12 Dibuje modelos de interruptor en el estilo de la figura 3-14 para una compuerta NOR CMOS de 2 entradas para todas las cuatro combinaciones de entrada.

3.13 Dibuje el diagrama de circuito, la tabla de funciones y el símbolo lógico para una compuerta OR CMOS en el estilo de la figura 3-19.

3.14 ¿Cuál tiene menos transistores, una compuerta no inversora o una compuerta inversora CMOS?

3.15 Nombre y dibuje los símbolos lógicos de cuatro diferentes compuertas CMOS de 4 entradas usando cada una 8 transistores.

3.16 El circuito en la figura X3.16(a) es un tipo de compuerta AND-OR-INVERSOR CMOS. Escriba una tabla de funciones para este circuito en el estilo de la figura 3.15(b) y su correspondiente diagrama lógico utilizando inversores y compuertas AND y OR.

3.17 El circuito en la figura X3.16(b) es un tipo de compuerta AND-OR-INVERSOR CMOS. Escriba la tabla de funciones para este circuito en el estilo de la figura 3.15(b) y su correspondiente diagrama lógico utilizando inversores y compuertas AND y OR.

3.18 ¿Cómo es que el perfume puede ser algo malo para los diseñadores digitales?

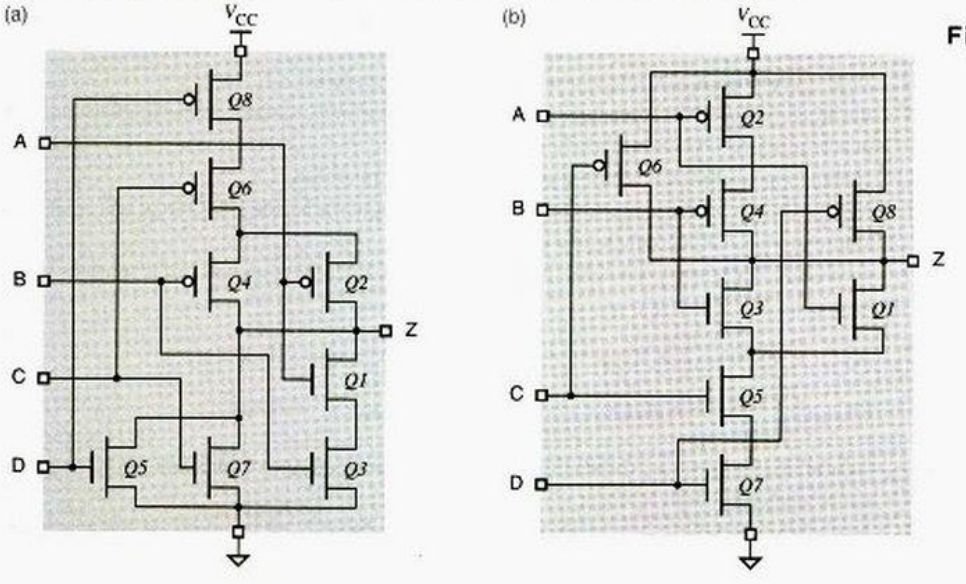


Figura X3.16

3.19 ¿Cuánto margen de ruido de CD de estado alto está disponible en un inversor CMOS cuya característica de transferencia bajo las condiciones del peor de los casos se parece a la de la figura 3-25? ¿Cuánto margen de ruido de CD de estado bajo se encuentra disponible? (Suponga umbrales estándar de 1.5 V y 3.5 V para BAJO y ALTO).

3.20 Haciendo uso de la hoja de datos en la tabla 3-3, determine los márgenes de ruido de CD del estado BAJO y del estado ALTO en el peor de los casos del 74HC00. Establezca cualquier suposición requerida por su respuesta.

3.21 La sección 3.5 define siete diferentes parámetros eléctricos para circuitos CMOS. Haciendo uso de la hoja de datos en la tabla 3-3, determine el valor del peor de los casos de cada uno de éstos para el 74HC00. Establezca cualquier suposición requerida por su respuesta.

3.22 Basándose en las convenciones y suposiciones de la sección 3.4, si la corriente para una salida de dispositivo se especifica como un número negativo, ¿la salida es una fuente de corriente o un sumidero de corriente?

3.23 Para cada una de las siguientes cargas resistivas, determine si las especificaciones de control de la salida del 74HC00 sobre el intervalo de operación comercial son excedidas. (Consulte la tabla 3-3, y haga uso de $V_{OHmin} = 2.4$ V y $V_{CC} = 5.0$ V.)

(a) 120 Ω para V_{CC} (b) 270 Ω para V_{CC} y 330 Ω para GND

(c) 1 kΩ para GND (d) 150 Ω para V_{CC} y 150 Ω para GND

(e) 100 Ω para V_{CC} (f) 75 Ω para V_{CC} y 150 Ω para GND

(g) 75 Ω para V_{CC} (h) 270 Ω para V_{CC} y 150 Ω para GND

3.24 A través del intervalo de niveles de entrada ALTO válidos, 2.0 – 5.0 V, ¿para qué nivel de entrada se esperaría que el 74FCT257T (tabla 3-3) consuma la mayor parte de la energía?

3.25 Determine el fan out de CD de estado BAJO y estado ALTO del 74FCT257T cuando controla entradas tipo 74LS00. (Consulte las tablas 3-3 y 3-12).

3.26 Haga la estimación de las resistencias de "encendido" de los transistores de salida de canal-*p* y canal-*n* del del 74FCT257T empleando la información de la tabla 3-3.

3.27 ¿En qué circunstancias es seguro permitir que una entrada CMOS sin utilizar quede flotando?

3.28 Explique el "latch-up" y las circunstancias en que ocurre.

3.29 Explique por qué no es una buena idea situar todos los capacitores de desacoplamiento en una esquina de un tablero de circuito impreso.

3.30 ¿Cuándo es importante tomarse de las manos con un amigo?

3.31 Nombre los dos componentes del retardo de compuerta lógica CMOS. ¿Cuál es el más afectado por la capacitancia de carga?

3.32 Determine la constante de tiempo *RC* para cada una de las siguientes combinaciones resistor-capacitor:

(a) $R = 100\ \Omega$, $C = 50$ pF (b) $R = 330\ \Omega$, $C = 150$ pF

(c) $R = 1$ kΩ, $C = 30$ pF (d) $R = 4.7$ kΩ, $C = 100$ pF

3.33 Aparte del retardo, ¿qué otra(s) característica(s) de un circuito CMOS se afecta(n) por la capacitancia de carga?

3.34 Explique la fórmula I_C en el pie de nota 5 de la tabla 3-3, en términos de los conceptos presentados en las secciones 3.5 y 3.6.

3.35 Es posible hacer funcionar dispositivos 74AC CMOS con una fuente de alimentación de 3.3 V. ¿Cuánta energía ahorrará esto por lo regular, en comparación con el funcionamiento a 5 V?

3.36 Un inversor de disparador Schmitt particular tiene $V_{IL,max} = 0.8$ V, $V_{IHmin} = 2.0$ V $V_{T+} = 1.6$ V y $V_{T-} = 1.3$ V. ¿Cuánta histéresis tiene?

3.37 ¿Por qué las salidas de tres estados se diseñan por lo regular para apagarse más rápido de lo que se encienden?

3.38 Discuta las ventajas y las desventajas de los resistores de disminución más grandes contra los más pequeños para las salidas CMOS de drenaje abierto o salidas TTL de colector abierto.

3.39 Un LED en particular tiene una caída de voltaje de aproximadamente 2.0 V en el estado "encendido" y requiere de alrededor de 5 mA de corriente para un brillo normal. Determine un valor apropiado para el resistor de arranque cuando el LED se encuentra conectado como se muestra en la figura 3-52.

3.40 ¿Cómo se modificará la respuesta para el problema 3.39 si el LED se conecta como se ilustra en la figura 3-53(a)?

3.41 Una función AND alambrada se obtiene simplemente al unir dos salidas de drenaje abierto o de colector abierto en conjunto, sin pasar a través de otro nivel de los circuitos de transistor. ¿Cómo, entonces, una función AND alambrada puede en realidad ser más lenta que una compuerta AND discreta? (*Sugerencia:* recuerde el título de una película de las Tortugas Ninja Adolescentes Mutantes.)

3.42 ¿Cuál familia lógica CMOS o TTL en este capítulo tiene la más fuerte capacidad de control de salida?

3.43 Resuma de manera concisa la diferencia entre las familias lógicas HC y HCT. La misma proposición concisa debería aplicarse a CA contra ACT.

3.44 ¿Por qué las especificaciones para los dispositivos FCT no incluyen parámetros como $V_{OL,maxC}$ que se aplican a las cargas CMOS, como se incluyen en las especificaciones de HCT y ACT?

3.45 ¿Cómo los dispositivos FCT-T reducen el consumo de energía en comparación con los dispositivos FCT?

3.46 ¿Cuántos diodos se requieren para una compuerta AND de diodo de n entradas?

3.47 Verdadero o falso: Una compuerta NOR TTL utiliza lógica de diodo.

3.48 ¿Las salidas TTL son más capaces para consumir corriente o para suministrar corriente?

3.49 Calcule el fan out máximo para cada uno de los siguientes casos de una salida TTL controlando múltiples entradas TTL. También indique cuánta capacidad de control en "exceso" se encuentra disponible en el estado BAJO o ALTO para cada caso.

(a) 74LS controlando a 74LS
(b) 74LS controlando a 74S
(c) 74S controlando a 74AS
(d) 74F controlando a 74S
(e) 74AS controlando a 74AS
(f) 74AS controlando a 74F
(g) 74ALS controlando a 74F
(h) 74AS controlando a 74ALS

3.50 ¿Cuál resistor disipa más energía, el de disminución para una entrada de compuerta NOR LS-TTL no utilizada, o el de arranque para una entrada de compuerta NAND LS-TTL no utilizada? Haga uso del mínimo valor de resistor disponible en cada caso.

3.51 ¿Cuál esperaría usted que fuera más rápido, una compuerta AND TTL o una compuerta AND-OR-INVERSOR TTL? ¿Por qué?

3.52 Describa el beneficio clave y la desventaja clave de los transistores Schottky en TTL.

3.53 Haciendo uso de la hoja de datos en la tabla 3-12, determine los márgenes de ruido de CD de estado BAJO y estado ALTO en el peor de los casos del 74LS00.

3.54 Las secciones 3.10.2 y 3.10.3 definen ocho diferentes parámetros eléctricos para circuitos TTL. Utilizando la hoja de datos en la tabla 3-12, determine el valor en el peor de los casos de cada uno de éstos para el 74LS00.

3.55 Para cada una de la siguientes cargas resistivas, determine si las especificaciones de control de salida del 74LS00 sobre el intervalo de operación comercial son excedidas. (Consulte la tabla 3-12, y haga uso de $V_{OLmax} = 0.5$ V y $V_{CC} = 5.0$ V.)

(a) 470 Ω para V_{CC} (b) 330 Ω para V_{CC} y 470 Ω para GND

(c) 10 kΩ para GND (d) 390 Ω para V_{CC} y 390 Ω para GND

(e) 600 Ω para V_{CC} (f) 510 Ω para V_{CC} y 510 Ω para GND

(g) 4.7 kΩ para GND (h) 220 Ω para V_{CC} y 330 Ω para GND

3.56 Calcule los márgenes de ruido de CD de estado BAJO y estado ALTO para cada uno de los casos siguientes de una salida TTL controlando una entrada CMOS compatible con TTL, o viceversa.

(a) 74HCT controlando a 74LS (b) 74VHCT controlando a 74AS

(c) 74LS controlando a 74HCT (d) 74S controlando a 74VHCT

3.57 Calcule el fan out para cada uno de los casos siguientes de una salida CMOS compatible con TTL que controla múltiples entradas en una familia lógica TTL. También indique cuánta capacidad de control "en exceso" se encuentra disponible en los estados BAJO o ALTO para cada caso.

(a) 74HCT controlando a 74LS (b) 74HCT controlando a 74S

(c) 74VHCT controlando a 74AS (d) 74VHCT controlando a 74LS

3.58 Para una capacitancia de carga y velocidad de transición determinadas, ¿cuál familia lógica en este capítulo tiene la disipación dinámica de energía de menor valor?

Ejercicios

3.59 Diseñe un circuito CMOS que tenga el comportamiento funcional mostrado en la figura X3.59. (*Sugerencia:* Sólo se requieren seis transistores.)

3.60 Diseñe un circuito CMOS que tenga el comportamiento funcional mostrado en la figura X3.60. (*Sugerencia:* Solamente se requieren seis transistores.)

3.61 Dibuje un diagrama de circuito, tabla de funciones y símbolo lógico en el estilo de la figura 3-19 para una compuerta CMOS con dos entradas A y B y una salida Z, donde Z = 1 si A = 0 y B = 1, y Z = 0 de otro modo. (*Sugerencia:* únicamente se necesitan seis transistores.)

Figura X3.59

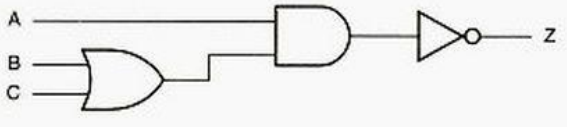
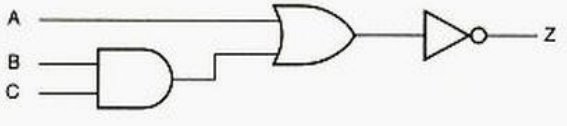


Figura X3.60



3.62 Dibuje un diagrama de circuito, tabla de funciones y símbolo lógico en el estilo de la figura 3-19 para una compuerta CMOS con dos entradas A y B y una salida Z, donde Z = 0 si A = 1 y B = 0, y Z = 1 en otro caso. (*Sugerencia:* Solamente se necesitan seis transistores.)

3.63 Dibuje una figura mostrando la estructura lógica de una compuerta NOR CMOS de 8 entradas, suponiendo que como máximo son prácticos circuitos de compuerta de 4 entradas. Haciendo uso de su conocimiento general de características CMOS, seleccione una estructura de circuito que minimice el retardo de propagación de la compuerta NOR para un área de silicio dada, y explique por qué esto es así.

3.64 Los diseñadores de circuitos de las familias CMOS compatibles con TTL presumiblemente podrían haber hecho la caída de voltaje a través del transistor "encendido" bajo carga en el estado ALTO tan pequeña como lo es en el estado BAJO, simplemente haciendo los transistores de canal-*p* más grandes. ¿Por qué supone usted que no se molestaron en hacer esto?

3.65 ¿Cuánta corriente y energía se "consumen" en la figura 3-32(b)?

3.66 Realice un cálculo detallado de V_{SAL} en las figuras 3-34 y 3-33. (*Sugerencia:* cree un equivalente de Thévenin para el inversor CMOS en cada figura.)

3.67 Considere el comportamiento dinámico de una salida CMOS controlando una carga capacitiva dada. Si la resistencia de la trayectoria de carga es el doble de la resistencia de la trayectoria de descarga, ¿el tiempo de ascenso es exactamente dos veces el tiempo de caída? Si no es así, ¿qué otros factores afectan los tiempos de transición?

3.68 Analice el tiempo de caída de la salida del inversor CMOS de la figura 3-37, con $R_L = 1\ k\Omega$ y $V_L = 2.5$ V. Compare su resultado con el de la sección 3.6.1 y explíquelo.

3.69 Repita el ejercicio 3.68 para tiempo de ascenso.

3.70 Suponiendo que los transistores en un buffer de tres estados CMOS FCT sean dispositivos perfectos con retardo cero de encendido-apagado, que conmuten a un umbral de entrada de 1.5 V, determine el valor de T_{PLZ} para el circuito de prueba y formas de onda de la figura 3-24. (*Sugerencia:* usted tiene que determinar el tiempo utilizando una constante de tiempo *RC*.) Explique la diferencia entre su resultado y las especificaciones de la tabla 3-3.

3.71 Repita el ejercicio 3.70 para t_{PHZ}.

3.72 Haciendo uso de las especificaciones de la tabla 3-6, haga la estimación de las resistencias "de encendido" de los transistores de canal-*p* y canal-*n* en la lógica CMOS de la serie 74AC.

3.73 Cree una matriz de $4 \times 4 \times 2 \times 2$ de los márgenes de ruido de CD en el peor de los casos para las siguientes situaciones de interfaz CMOS: una salida (HC, HCT, VHC o VHCT) controlando una entrada (HC, HCT, VHC o VHCT) con una carga (CMOS, TTL) en el estado (BAJO, ALTO); la figura X3.73 ilustra el caso. (*Sugerencias:* existen 64 diferentes combinaciones, pero muchas proporcionan resultados idénticos. Algunas combinaciones producen márgenes negativos.)

Entrada

Salida	HC		HCT		VHC		VHCT	
HC	CL	TL	CL	TL	CL	TL	CL	TL
	CH	TH	CH	TH	CH	TH	CH	TH
HCT	CL	TL	CL	TL	CL	TL	CL	TL
	CH	TH	CH	TH	CH	TH	CH	TH
VHC	CL	TL	CL	TL	CL	TL	CL	TL
	CH	TH	CH	TH	CH	TH	CH	TH
VHCT	CL	TL	CL	TL	CL	TL	CL	TL
	CH	TH	CH	TH	CH	TH	CH	TH

Clave:
CL = carga CMOS, BAJO
CH = carga CMOS, ALTO
TL = carga TTL, BAJO
TH = carga TTL, ALTO

Figura X3.73

3.74 Haciendo uso de la figura 3-85, determine los márgenes de ruido de CD para la lógica CMOS de 5 V controlando CMOS de 3.3 V tolerante a 5 V, y viceversa.

3.75 Con base en la figura 3.85, determine los márgenes de ruido de CD para CMOS de 3.3 V controlando CMOS de 2.5 V tolerante a 3.3 V, y viceversa.

3.76 En el ejemplo de LED de la sección 3.7.5, un diseñador elige un valor de resistor de 300 Ω y encuentra que la compuerta de drenaje abierto era capaz de mantener su salida a 0.1 V mientras controla el LED. ¿Cuánta corriente fluye a través del LED y cuánta energía disipa el resistor de arranque en este caso?

3.77 Considere un contador binario CMOS de 8 bits (sección 8.4) temporizado a 16 MHz. Con el propósito de calcular la disipación dinámica de energía en el contador, ¿cuál es la frecuencia de transición del bit menos significativo? ¿Y del bit más significativo? Para el propósito de determinar la disipación dinámica de energía de los ocho bits de salida, ¿qué frecuencia debería utilizarse?

3.78 Empleando únicamente compuertas AND y NOR, dibuje un diagrama lógico para la función lógica realizada por el circuito en la figura 3-55.

3.79 Calcule el voltaje de salida aproximado en Z en la figura 3-56, suponiendo que las compuertas sean CMOS serie HCT.

3.80 Vuelva a dibujar el circuito de un buffer de tres estados CMOS en la figura 3-48 empleando transistores reales en lugar de símbolos NAND, NOR e inversores. ¿Puede usted encontrar un circuito para la misma función que requiera un número total más pequeño de transistores? Si es así, dibújelo.

3.81 Modifique el circuito de buffer de tres estados CMOS en la figura 3-48 de modo que la salida se encuentre en el estado Hi-Z cuando la entrada de habilitación esté en nivel ALTO. El circuito modificado no debería requerir de más transistores que el original.

3.82 Haciendo uso de la información de la tabla 3-3, haga la estimación de cuánta corriente puede fluir a través de cada terminal de salida, si las salidas de dos diferentes 74HC00 se encuentran en pugna.

3.83 Demuestre que para un voltaje de alimentación dado, una especificación I_{CCD} tipo FCT puede derivarse de una especificación C_{PD} tipo HCT/ACT, y viceversa.

3.84 Si tanto *VZ* como *V_B* en la figura 3-65(b) sea 4.6 V, ¿podemos obtener $V_C = 5.2$ V? Explique por qué.

3.85 Modifique el programa de la tabla 3-10 para dar cuenta de la corriente de fuga en el estado apagado OFF.

3.86 Suponiendo condiciones "ideales", ¿cuál es el voltaje mínimo que será reconocido como un ALTO en la compuerta NAND TTL de la figura 3-75 con una entrada BAJO y la otra ALTO?

3.87 Suponiendo condiciones "ideales", ¿cuál es el voltaje máximo que será reconocido como un BAJO en la compuerta NAND TTL de la figura 3-75 con ambas entradas en ALTO?

3.88 Encuentre una parte TTL comercial que pueda suministrar 40 mA en el estado ALTO. ¿Cuál es su aplicación?

3.89 ¿Qué ocurre si se intenta controlar un LED con su cátodo aterrizado y su ánodo conectado a una salida de totem-pole TTL, análoga a la figura 3-53 para CMOS?

3.90 ¿Qué ocurre si se intenta controlar un relevador de 12 volts con una salida de totem-pole TTL?

3.91 Suponga que un resistor de arranque simple para +5 V se utiliza para proporcionar una fuente lógica constante 1 para 15 diferentes entradas 74LS00. ¿Cuál es el valor máximo del este resistor? ¿Cuánto margen de ruido de CD de estado ALTO se está proporcionando en este caso?

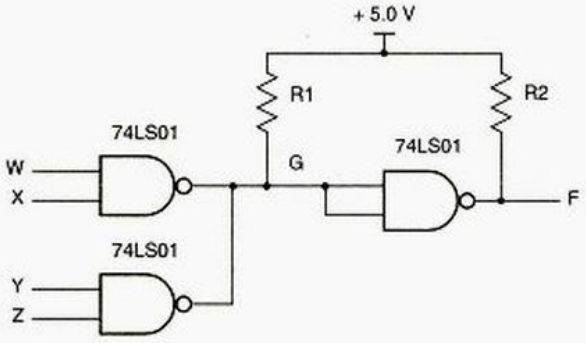


Figura X3.92

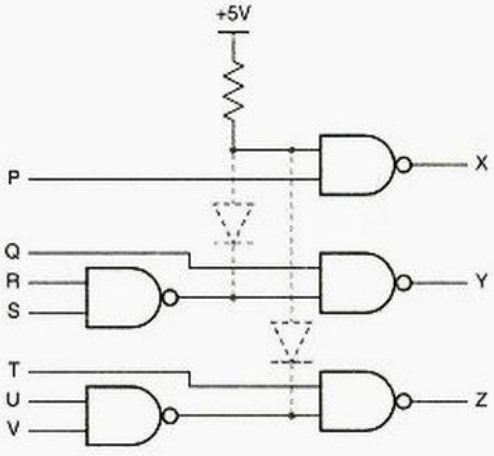


Figura X3.95

3.92 El circuito en la figura X3-92 utiliza compuertas NAND de colector abierto para realizar "lógica alambrada". Escriba una tabla de verdad para la señal de salida F y, si se ha leído la sección 4.2, una expresión lógica para F como una función de las entradas del circuito.

3.93 ¿Cuál es el valor máximo permisible para *R1* en la figura X3-92? Suponga que se requiere de un margen de ruido de estado ALTO de 0.7 V. El 74LS01 tiene las especificaciones mostradas en la columna 74LS de la tabla 3-11, excepto que I_{OHmax} es 100 μA, una corriente de fuga que fluye *dentro* de la salida en el estado ALTO.

3.94 Suponga que la señal de salida F en la figura X3-92 controla las entradas de dos inversores 74S04. Calcule los valores mínimo y máximo permisibles de *R2* suponiendo que se requiere de un margen de ruido de estado ALTO de 0.7 V.

3.95 Un diseñador lógico encuentra un problema en cierta función del circuito después de que el mismo ha sido liberado para la producción y construido 1000 copias del mismo. Un segmento del circuito se ilustra en la figura X3.95 con líneas continuas; todas las compuertas son compuertas 74Ls00. El diseñador lógico arregla el problema al agregar los dos diodos mostrados con líneas discontinuas. ¿Qué es lo que hacen los diodos? Describa tanto los efectos lógicos de este cambio en la función del circuito, como los efectos eléctricos en los márgenes de ruido del circuito.

3.96 Un 74LS125 es un buffer con una salida de tres estados. Cuando se habilita, la salida puede consumir 24 mA en el estado BAJO y suministrar 2.6 mA en el estado ALTO. Cuando se deshabilita, la salida tiene una corriente de fuga de $\pm$ 20 μA (el signo depende del voltaje de salida: positivo si la salida se lleva a ALTO por otros dispositivos, negativo si es BAJO). Suponga que se diseña un sistema con múltiples módulos conectados a un bus, donde cada módulo tiene un solo 74LS125 para controlar el bus y un 74LS04 para recibir información en el bus. ¿Cuál es el número máximo de módulos que pueden conectarse al bus sin exceder las especificaciones del 74LS125?

3.97 Repita el ejercicio 3.97, suponiendo esta vez que se conecta un resistor de arranque simple desde el bus a +5 V para garantizar que el bus se encuentre en ALTO cuando ningún dispositivo esté controlándolo. Calcule el máximo valor posible del resistor de arranque, así como el número de módulos que pueden conectarse al bus.

3.98 Encuentre el diseño del circuito en un libro de datos TTL para una compuerta de triple estado real, y explique cómo funciona.

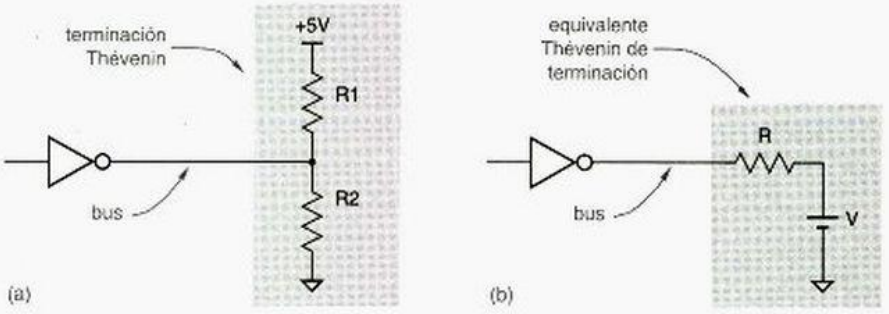


Figura X3.99

3.99 Una *terminación Thévenin* para un bus de tres estados o de colector abierto tiene la estructura mostrada en la figura X3.99(a). La idea es que, al seleccionar valores apropiados de *R1* y *R2*, un diseñador pueda obtener un circuito equivalente a la terminación en (b) para cualesquiera valores deseados de *V* y *R*. El valor de *V* determina el voltaje en el bus cuando ningún dispositivo lo está controlando, y el valor de *R* se selecciona para satisfacer la impedancia característica del bus para propósitos de línea de transmisión (sección 11.4). Para cada uno de los pares siguientes de *V* y *R*, determine los valores requeridos de *R1* y *R2*.

(a) $V = 2.75, R = 148.5$ (b) $V = 2.7, R = 180$

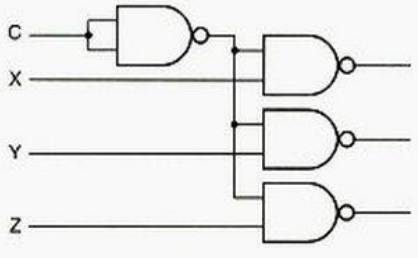
(c) $V = 3.0, R = 130$ (d) $V = 2.5, R = 75$

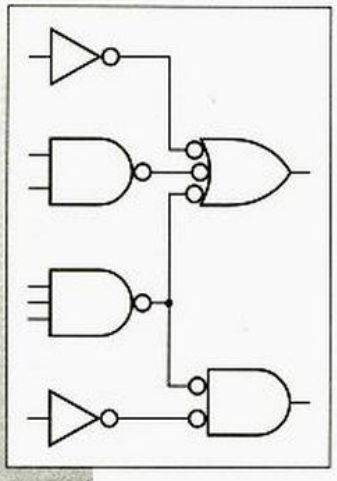
3.100 Para cada uno de los pares *R1* y *R2* en el ejercicio 3.99, determine si la terminación puede ser apropiadamente controlada por una salida de tres estados en cada uno de las siguientes familias lógicas: 74LS, 74S, 74ACT. Para una operación apropiada, las especificaciones I_{OL} e I_{OH} de la familia no deben ser excedidas cuando $V_{OL} = V_{OLmax}$ y $V_{OH} = V_{OHmin}$, respectivamente.

3.101 Utilizando las gráficas en un libro de datos TTL, desarrolle algunas reglas prácticas para corregir o disminuir la especificación de retardo de propagación máximo de LS-TTL bajo condiciones no óptimas de voltaje de alimentación, temperatura y carga.

3.102 Determine la disipación total de energía del circuito en la figura X3.102 como función de la frecuencia de transición *f* para dos casos: a) utilizando compuertas 74LS; b) haciendo uso de compuertas 74HC. Suponga que la capacitancia de entrada es de 3 pF para una compuerta TTL y de 7 pF para una compuerta CMOS, que una compuerta 74LS tiene una capacitancia de disipación de energía interna de 20 pF y que existen 20 pF adicionales de capacitancia alambrada de pérdida en el circuito. También suponga que las entradas X, Y y Z están siempre en ALTO, y que la entrada C se encuentra controlada por una onda cuadrada de nivel CMOS con frecuencia *f*. Otra información que usted necesitará para este problema puede hallarse en las tablas 3-5 y 3-11. Establezca cualquier otra suposición que haga. ¿A qué frecuencia el circuito TTL disipa menos energía que el circuito CMOS?

Figura X3.102





capítulo 4

Principios de diseño lógico-combinacional

Los circuitos lógicos se clasifican en dos tipos, "combinacional" y "secuencial". Un *circuito lógico combinacional* es aquél cuyas salidas dependen solamente de sus entradas actuales. La perilla selectora de canales de un receptor de televisión B/N es como un circuito combinacional: su "salida" selecciona un canal basada solamente en la posición actual de la perilla ("entrada").

Las salidas de un *circuito lógico secuencial* dependen no sólo de sus entradas actuales, sino también de la secuencia anterior de las entradas, quizás arbitrariamente lejos en el pasado. El selector de canales, que se controla mediante los botones "hacia arriba" y "hacia abajo" de un televisor o de una videocasetera, es un circuito secuencial. La selección de canal depende de la secuencia anterior de las selecciones hacia arriba/hacia abajo, por lo menos desde que se comenzó a ver 10 horas antes, y quizás desde cuando se encendió el dispositivo. Estos circuitos se analizan en los capítulos 7 y 8.

Un circuito combinacional puede contener una cantidad arbitraria de compuertas lógicas e inversores pero no lazos de retroalimentación. Un *lazo de retroalimentación* es la trayectoria de una señal en un circuito que permite que la salida de una compuerta se propague de regreso hacia la entrada de esa misma compuerta; un lazo de esa naturaleza generalmente crea un comportamiento secuencial en el circuito.

En el *análisis* de un circuito combinacional comenzamos con un diagrama lógico y procedemos hasta una descripción formal de la función que realiza el circuito, tal como una tabla de verdad o una expresión lógica. En la *síntesis* hacemos lo contrario, comenzamos con una descripción formal y procedemos hasta un diagrama lógico.

SÍNTESIS VERSUS DISEÑO

El diseño de circuitos lógicos es un superconjunto de la síntesis, puesto que en un problema de diseño real por lo regular comenzamos con una descripción informal (de palabra o pensamiento) del circuito. Con frecuencia la parte más desafiante y creativa del diseño es formalizar la descripción del circuito, definiendo las señales de entrada y salida del circuito y especificando su comportamiento funcional por medio de tablas de verdad y ecuaciones. Una vez que hemos creado la descripción formal del circuito, podemos habitualmente seguir un procedimiento de síntesis "práctico" para obtener un diagrama lógico para un circuito con el comportamiento funcional requerido. El material en las primeras cuatro secciones de este capítulo es la base para los procedimientos "prácticos" o "para ponerse a trabajar", ya sea si el trabajo se hace a mano o por computadora. Las últimas dos secciones describen lenguajes de diseño reales, ABEL y VHDL. Cuando creamos un diseño utilizando uno de estos lenguajes, un programa de computadora puede realizar los pasos de la síntesis por nosotros. En capítulos posteriores encontraremos muchos ejemplos del proceso real de diseño.

Los circuitos combinacionales pueden tener una o más salidas. La mayor parte de las técnicas de análisis y síntesis se extienden de una manera obvia desde los circuitos que tienen una salida hasta los circuitos con varias salidas (por ejemplo, "Repita estos pasos para cada salida"). También señalaremos cómo se pueden aplicar algunas técnicas de una manera no tan obvia para obtener más eficacia en el caso de las salidas múltiples.

El propósito de este capítulo es darle a usted un fundamento teórico sólido para el análisis y la síntesis de los circuitos lógicos combinacionales, un fundamento que más tarde será doblemente importante cuando estudiemos los circuitos secuenciales. Aunque en la actualidad la mayoría de los procedimientos de análisis y síntesis de este capítulo están automatizados (mediante herramientas de diseño asistidas por computadora), usted necesitará un conocimiento básico de los fundamentos para utilizar las herramientas y para encontrar lo que está mal cuando obtenga resultados inesperados o indeseables.

Una vez que tiene los fundamentos, lo siguiente es comprender cómo se pueden expresar y analizar las funciones combinacionales mediante el uso de lenguajes de descripción de hardware (HDL). Así, las últimas dos secciones de este capítulo presentan las características básicas de ABEL y VHDL, los cuales emplearemos para diseñar toda clase de circuitos lógicos en todo el texto.

Antes de iniciar el estudio de los circuitos lógicos combinacionales, debemos presentar el álgebra de conmutación, la herramienta matemática fundamental para analizar y sintetizar circuitos lógicos de todas clases.

4.1 Álgebra de conmutación

álgebra booleana

Las técnicas de análisis formal para los circuitos digitales tienen sus raíces en el trabajo de un matemático inglés, George Boole. En 1854, él inventó un sistema algebraico de dos valores, que ahora se conoce con el nombre de *álgebra de conmutación*, para "dar expresión ...a las leyes fundamentales del razonamiento en el lenguaje simbólico del Cálculo".

Al utilizar este sistema, un filósofo, un especialista en lógica o un habitante del planeta Vulcano es capaz de formular instrucciones que sean verdaderas o falsas, combinarlas para hacer nuevas instrucciones y determinar la verdad o falsedad de las nuevas instrucciones. Por ejemplo, si estamos de acuerdo en que "la gente que no ha estudiado este material o es fracasada o no es intelectual" y "ningún diseñador de computadoras es un fracasado", entonces podremos responder las siguientes preguntas "si eres un diseñador de computadoras intelectual, entonces ¿ya has estudiado esto?"

Mucho después de Boole, en 1938, el investigador de los Laboratorios Bell, Claude E. Shannon, demostró cómo se podía adaptar el álgebra de conmutación para analizar y describir el comportamiento de los circuitos que se construyeron a partir de relevadores. Los relevadores fueron los elementos lógicos digitales que se utilizaron con mayor frecuencia en ese momento. En la obra *Álgebra de conmutación* de Shannon, el estado de los contactos (platinos) de un relevador, abierto o cerrado, está representado por una variable X, que puede tener uno de dos valores posibles, 0 o 1. En las tecnologías lógicas de la actualidad, estos valores corresponden a una amplia variedad de condiciones físicas: voltaje ALTO o BAJO, luz apagada o encendida, capacitor descargado o cargado, fusible fundido o intacto, y así sucesivamente, como indicamos en la tabla 3-1 de la página 81.

álgebra de conmutación

En el resto de esta sección desarrollaremos directamente el álgebra de conmutación, para tal fin utilizaremos los "principios primarios" y lo que ya sabemos acerca del comportamiento de los elementos lógicos (compuertas e inversores). Para tener una mayor idea histórica y/o matemática de este material, consulte las referencias.

4.1.1 Axiomas

En el álgebra de conmutación utilizamos una variable simbólica, p. ej. la X, para representar la condición de una señal lógica. Una señal lógica se encuentra en una de dos posibles condiciones: bajo o alto, apagado o encendido, etcétera, dependiendo de la tecnología. Decimos que X tiene el valor "0" para una de estas condiciones y "1" para la otra.

Por ejemplo, en los circuitos lógicos CMOS y TTL del capítulo 3, la *convención de lógica positiva* indica que debemos asociar el valor "0" con un voltaje BAJO y "1" con un voltaje ALTO. La *convención de lógica negativa* hace la asociación opuesta: 0 = ALTO y 1 = BAJO. Sin embargo, la selección de la lógica positiva o negativa no tiene efecto sobre nuestra capacidad de desarrollar una descripción algebraica consistente del comportamiento del circuito; solamente afecta los detalles de la abstracción física-algebraica, como lo explicaremos posteriormente en nuestro análisis de la "dualidad". Por el momento, ignoraremos las realidades físicas de los circuitos lógicos y asumiremos que funcionen directamente sobre los símbolos lógicos 0 y 1.

convención de lógica positiva

convención de lógica negativa

Los *axiomas* (o *postulados*) de un sistema matemático son un conjunto mínimo de definiciones básicas que suponemos verdaderas, a partir de estas definiciones se obtiene información adicional del sistema. Los primeros dos axiomas del álgebra de conmutación expresan la "abstracción digital" al establecer formalmente que una variable X solamente puede tomar uno de dos valores:

axioma
postulado

$$(A1) \quad X = 0 \quad \text{si } X \neq 1 \qquad (A1') \quad X = 1 \quad \text{si } X \neq 0$$

Note que establecimos estos axiomas como un par, la única diferencia entre A1 y A1′ es el intercambio de los símbolos 0 y 1. Ésta es una característica de todos los axiomas del

álgebra de conmutación y es la base del principio de "dualidad" que estudiaremos más tarde.

complemento prima (′)

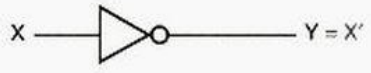
En la sección 3.3.3 mostramos el diseño de un inversor, que es un circuito lógico. En un inversor el nivel de la señal de salida es opuesto (o *complemento*) al nivel de la señal de entrada. Utilizamos el signo de *prima* (′) para indicar una función inversora. Es decir, si una variable X indica la señal de entrada del inversor, entonces X′ indica el valor de la señal de salida del inversor. Esta notación se especifica formalmente en el segundo par de axiomas:

(A2) Si X = 0, entonces X′ = 1 (A2′) si X = 1, entonces X′ = 0

operador algebraico operación NOT

Como se muestra en la figura 4-1, la salida de un inversor que tiene una señal de entrada X, puede tener una designación arbitraria, por ejemplo Y. Sin embargo, algebraicamente escribimos Y = X′ para expresar que "la señal Y siempre tiene el valor opuesto de la señal X". El símbolo prima (′) representa un *operador algebraico*, y X′ es una *expresión*, la cual se lee como "X prima" o "NOT X". Este uso es análogo al que hemos aprendido en los lenguajes de programación. En estos lenguajes si J es una variable entera, entonces ¬J es una expresión cuyo valor es 0 − J. Aunque al parecer esto no tiene mucha importancia, aprenderemos que la distinción entre los nombres de las señales (X, Y), las expresiones (X′) y las ecuaciones (Y = X′) es muy importante cuando estudiamos los estándares de documentación y las herramientas de software para el diseño lógico. En los diagramas lógicos de este libro mantenemos esta distinción escribiendo nombres de señal en negro y las expresiones en color gris.

Figura 4-1
Identificación de la señal y notación algebraica para un inversor.



multiplicación lógica punto de multiplicación (·)

En la sección 3.3.6 demostramos cómo construir una compuerta AND CMOS de 2 entradas, un circuito cuya salida es 1 solamente si sus dos entradas son 1. La función de una compuerta AND de 2 entradas se denomina en ocasiones *multiplicación lógica*, y se representa algebraicamente con un *punto de multiplicación* (·). Es decir, una compuerta AND con entradas X y Y tiene una señal de salida cuyo valor es X · Y, como se muestra en la figura 4-2(a). Algunos autores, especialmente los matemáticos y los especialistas en

NOTA SOBRE LA NOTACIÓN

Algunos autores también utilizan la notación $\overline{X}$, ~X y ¬X para indicar el complemento de X. La notación de barra superior ($\overline{X}$) es probablemente la más ampliamente utilizada y la que tiene mejor aspecto tipográfico. Sin embargo, utilizamos la notación con prima para conseguir que usted emplee la escritura de expresiones lógicas en una sola línea de texto sin la barra superior más gráfica, y para forzarlo a poner entre paréntesis subexpresiones complejas complementadas; porque esto tendrá usted que hacerlo cuando utilice los HDL y otras herramientas.

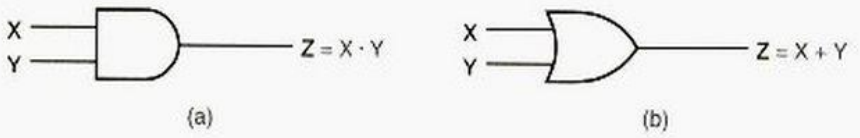


Figura 4-2
Identificación de señal y notación algebraica: a) compuerta AND; b) compuerta OR.

lógica, indican la multiplicación lógica con una cuña ($X \vee Y$). En la práctica seguiremos el estándar de la ingeniería utilizando el punto ($X \cdot Y$). Cuando analicemos los lenguajes de descripción de hardware (HDLs, por sus siglas en inglés), encontraremos otros símbolos diversos que se utilizan para indicar la misma operación.

También describimos en la sección 3.3.6 cómo construir una compuerta OR CMOS de 2 entradas, un circuito cuya salida es 1 si cualquiera de sus entradas es 1. La función de una compuerta OR con 2 entradas se conoce también como *suma lógica*, la cual se simboliza algebraicamente por un signo de suma (+). Una compuerta OR con entradas X y Y tiene una salida de señal cuyo valor es $X + Y$, como se muestra en la figura 4-2(b). Algunos autores denotan la suma lógica con una cuña ($X \wedge Y$), pero seguiremos la práctica estándar de la ingeniería que utiliza el signo de suma ($X + Y$). Una vez más, pueden usarse otros símbolos en los HDLs. Por convención, en una expresión lógica que involucre tanto la multiplicación como la suma, la multiplicación tiene *precedencia*, como sucede en las expresiones con enteros en los lenguajes convencionales de programación. Es decir, la expresión $W \cdot X + Y \cdot Z$ es equivalente a $(W \cdot X) + (Y \cdot Z)$.

suma lógica

precedencia

Los últimos tres pares de axiomas establecen las definiciones formales de las operaciones AND y OR, además, indican la salida que produce cada compuerta para cada posible combinación de entrada:

operación AND
operación OR

(A3)	$0 \cdot 0 = 0$	(A3′)	$1 + 1 = 1$
(A4)	$1 \cdot 1 = 1$	(A4′)	$0 + 0 = 0$
(A5)	$0 \cdot 1 = 1 \cdot 0 = 0$	(A5′)	$1 + 0 = 0 + 1 = 1$

Los cinco pares de axiomas, A1–A5 y A1′–A5′, definen completamente el álgebra de conmutación. Todos los otros hechos acerca del sistema se demuestran utilizando estos axiomas como punto de partida.

ESPERE ("YUXTAPONGA") UN MINUTO ...

Los textos antiguos utilizan la *yuxtaposición* simple (XY) para indicar la multiplicación lógica, pero nosotros no. En general, la yuxtaposición es una notación clara solamente cuando los nombres de señales se encuentran limitados por un solo carácter. De otro modo, ¿XY sería un producto lógico o un nombre de señal de dos caracteres? Los nombres de variable de un carácter son comunes en el álgebra, pero en problemas reales de diseño digital preferimos utilizar nombres de señal con caracteres múltiples que nos digan algo. De esta manera, necesitamos un separador entre nombres, y el separador puede sencillamente ser un punto de multiplicación en vez de un espacio. El equivalente HDL del punto de multiplicación (que es a menudo * o &) es absolutamente requerido cuando se escriben fórmulas lógicas en un lenguaje de descripción de hardware.

Tabla 4-1
Teoremas del álgebra de conmutación con una variable.

(T1)	$X + 0 = X$	(T1′)	$X \cdot 1 = X$	(Identidades)
(T2)	$X + 1 = 1$	(T2′)	$X \cdot 0 = 0$	(Elementos nulos)
(T3)	$X + X = X$	(T3′)	$X \cdot X = X$	(Potencia idéntica)
(T4)	$(X')' = X$			(Involución)
(T5)	$X + X' = 1$	(T5′)	$X \cdot X' = 0$	(Complementos)

4.1.2 Teoremas de una sola variable

Durante el análisis o la síntesis de los circuitos lógicos, con frecuencia escribimos expresiones algebraicas que caracterizan el comportamiento deseado o real de un circuito. Los *teoremas* del álgebra de conmutación son enunciados, que a nuestro saber siempre son ciertos, los cuales nos permiten manipular las expresiones algebraicas para aplicar un análisis más sencillo o realizar una síntesis más eficaz de los circuitos correspondientes. Por ejemplo, el teorema $X + 0 = X$ nos permite sustituir $X + 0$ cada vez que se presente en una expresión con la X.

teorema

La tabla 4-1 enumera los teoremas del álgebra de conmutación que involucran una sola variable X. ¿Cómo sabemos que estos teoremas son verdaderos? Podemos demostrarlos por nosotros mismos o bien aceptar la palabra de alguien que lo haya hecho. Como somos estudiantes de nivel universitario, aprenderemos a demostrarlos.

La demostración de la mayor parte de los teoremas en el álgebra de conmutación es extremadamente sencilla, si se aplica una técnica llamada de *inducción perfecta*. El axioma A1 es la clave para esta técnica: ya que una variable de conmutación solamente puede tomar dos valores diferentes, 0 y 1. El teorema se demuestra involucrando una sola variable X, probando que es verdadero tanto para $X = 0$ como para $X = 1$. Por ejemplo, para demostrar el teorema T1, hagamos dos sustituciones:

inducción perfecta

$[X = 0]$	$0 + 0 = 0$	verdadero, de acuerdo al axioma A4′
$[X = 1]$	$1 + 0 = 1$	verdadero, de acuerdo al axioma A5′

Todos los teoremas de la tabla 4-1 se demuestran aplicando la inducción perfecta, lo mismo tiene que hacer en los problemas propuestos 4.2 y 4.3.

4.1.3 Teoremas de dos y tres variables

Los teoremas del álgebra de conmutación con dos o tres variables se enumeran en la tabla 4.2. Cada uno de estos teoremas se demuestra fácilmente por inducción perfecta, evaluando la instrucción del teorema para las cuatro posibles combinaciones de X y Y, o las ocho posibles combinaciones de X, Y y Z.

Los primeros dos pares de teoremas se relacionan con la conmutatividad y la asociatividad de la suma y multiplicación lógicas, por lo tanto son idénticos a las leyes conmutativa y asociativa de la suma y multiplicación de números enteros y reales. Tomados en conjunto, indican que la agrupación u orden de los términos en una suma lógica o producto lógico es irrelevante. Por ejemplo, desde un punto de vista estrictamente algebraico, una expresión como $W \cdot X \cdot Y \cdot Z$ es ambigua; debería escribirse como $(W \cdot (X \cdot (Y \cdot Z)))$ o $(((W \cdot X) \cdot Y) \cdot Z)$ o $(W \cdot X) \cdot (Y \cdot Z)$ (consulte el ejercicio 4.34). Pero los teoremas nos

Tabla 4-2 Teoremas del álgebra de conmutación con dos o tres variables.

(T6)	$X + Y = Y + X$	(T6′)	$X \cdot Y = Y \cdot X$	(Conmutatividad)
(T7)	$(X + Y) + Z = X + (Y + Z)$	(T7′)	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	(Asociatividad)
(T8)	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$	(T8′)	$(X + Y) \cdot (X + Z) = X + Y \cdot Z$	(Distributividad)
(T9)	$X + X \cdot Y = X$	(T9′)	$X \cdot (X + Y) = X$	(Cubierta)
(T10)	$X \cdot Y + X \cdot Y' = X$	(T10′)	$(X + Y) \cdot (X + Y') = X$	(Combinación)
(T11)	$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$			(Consenso)
(T11′)	$(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$			

indican que la forma ambigua de la expresión está bien, puesto que obtenemos los mismos resultados en cualquier caso. Incluso podemos modificar el orden de las variables (por ejemplo, $X \cdot Z \cdot Y \cdot W$) y obtener los mismos resultados.

Aunque este análisis parezca trivial es muy importante porque establece los fundamentos teóricos que rigen el uso de las compuertas lógicas que tienen más de dos entradas. Definimos · y + como *operadores binarios*: operadores que combinan *dos* variables. Sin embargo, en la práctica utilizamos compuertas AND y OR con 3 o 4 entradas (o más). Los teoremas nos indican que las entradas de las compuertas se pueden conectar en cualquier orden; de hecho, muchos programas que definen la colocación de los componentes en las tarjetas de circuito impreso y programas de diseño ASIC toman ventaja de este hecho. Podemos utilizar una compuerta con n entradas o varias compuertas con 2 entradas (n–1) en forma intercambiable, aunque el retardo de propagación y el costo sean probablemente mayores que en el caso de las compuertas múltiples de 2 entradas.

operador binario

El teorema T8 es idéntico a la ley distributiva de los números enteros y reales, es decir, la multiplicación lógica se distribuye sobre la suma lógica. En consecuencia, podemos "expandir" una expresión para obtener una suma de productos, como en el ejemplo que sigue:

$$V \cdot (W + X) \cdot (Y + Z) = V \cdot W \cdot Y + V \cdot W \cdot Z + V \cdot X \cdot Y + V \cdot X \cdot Z$$

Sin embargo, el álgebra de conmutación también tiene una propiedad poco familiar, en la cual el inverso es verdadero, donde la suma lógica se distribuye sobre la multiplicación lógica, como se demuestra mediante el teorema T8′. De este modo, también "expandimos" una expresión para obtener una forma de producto de sumas:

$$(V \cdot W \cdot X) + (Y \cdot Z) = (V + Y) \cdot (V + Z) \cdot (W + Y) \cdot (W + Z) \cdot (X + Y) \cdot (X + Z)$$

Los teoremas T9 y T10 se utilizan ampliamente en la minimización de las funciones lógicas. Por ejemplo, si la subexpresión $X + X \cdot Y$ aparece en una expresión lógica, el *teorema de cubierta* T9 nos indica que solamente necesitamos incluir X en la expresión; se dice que X *cubre a* $X \cdot Y$. El *teorema de combinación* T10 nos indica lo siguiente: si la subexpresión $X \cdot Y + X \cdot Y'$ aparece en una expresión, entonces podemos reemplazarla con X. Puesto que Y debe ser 0 o 1, de cualquier manera la subexpresión original es 1 si y sólo si X es 1.

teorema de cubierta
cubierta
teorema de combinación

Aunque es posible demostrar fácilmente T9 por inducción perfecta, la veracidad de T9 es más obvia si la demostramos utilizando los otros teoremas que ya hemos demostrado hasta ahora:

$$\begin{aligned} X + X \cdot Y &= X \cdot 1 + X \cdot Y && \text{(de acuerdo con T1')} \\ &= X \cdot (1 + Y) && \text{(de acuerdo con T8)} \\ &= X \cdot 1 && \text{(de acuerdo con T2)} \\ &= X && \text{(de acuerdo con T1')} \end{aligned}$$

Igualmente, se pueden utilizar los teoremas adicionales para demostrar T10, donde el paso clave es utilizar T8 para volver a escribir el lado izquierdo como $X \cdot (Y + Y')$.

teorema de consenso
consenso

El teorema T11 se conoce como el *teorema de consenso*. El término $Y \cdot Z$ se conoce como el *consenso* de $X \cdot Y$ y $X' \cdot Z$. La idea es que si $Y \cdot Z$ es 1, entonces $X \cdot Y$ o $X' \cdot Z$ también deben ser 1, puesto que Y y Z son ambos 1 y X o X' deben ser 1. Así, el término $Y \cdot Z$ es redundante y se elimina del lado derecho de T11. El teorema de consenso tiene dos aplicaciones importantes. Se emplea para eliminar ciertos riesgos de temporización en circuitos lógicos combinacionales, como lo veremos en la sección 4.5. Y también forma el fundamento del método de consenso iterativo para hallar los implicantes primos (véase la sección de Referencias).

En todos los teoremas es posible reemplazar cada variable con una expresión lógica arbitraria. Un reemplazo simple es asociar una o más variables:

$$(X + Y') + Z' = X + (Y' + Z') \quad \text{(basado en T7)}$$

Pero expresiones más complejas también pueden ser sustituidas:

$$(V' + X) \cdot (W \cdot (Y' + Z)) + (V' + X) \cdot (W \cdot (Y' + Z))' = V' + X \quad \text{(basado en T10)}$$

4.1.4 Teoremas de *n* variables

Varios teoremas importantes, numerados en la tabla 4-3, son verdaderos para un número arbitrario de variables, *n*. La mayoría de estos teoremas se demuestran utilizando un

inducción finita
paso fundamental
paso de inducción

Así, el teorema es verdadero para todos los valores finitos de *n*.

Los *teoremas de DeMorgan* (T13 y T13') son probablemente los más comúnmente utilizados entre todos los teoremas del álgebra de conmutación. El teorema T13 nos dice que una compuerta AND de *n* entradas cuya salida está complementada es equivalente a

$$\begin{aligned} X + X + X + \cdots + X &= X + (X + X + \cdots + X) && (i + 1 \text{ X's en cada lado}) \\ &= X + (X) && (\text{si T12 es verdadero para } n = i) \\ &= X && \text{(de acuerdo con T3)} \end{aligned}$$

Así, el teorema es verdadero para todos los valores finitos de *n*.

teoremas de DeMorgan

Los *teoremas de DeMorgan* (T13 y T13') son probablemente los más comúnmente utilizados entre todos los teoremas del álgebra de conmutación. El teorema T13 nos dice que una compuerta AND de *n* entradas cuya salida está complementada es equivalente a

Tabla 4-3 Teoremas del álgebra de conmutación con *n* variables.

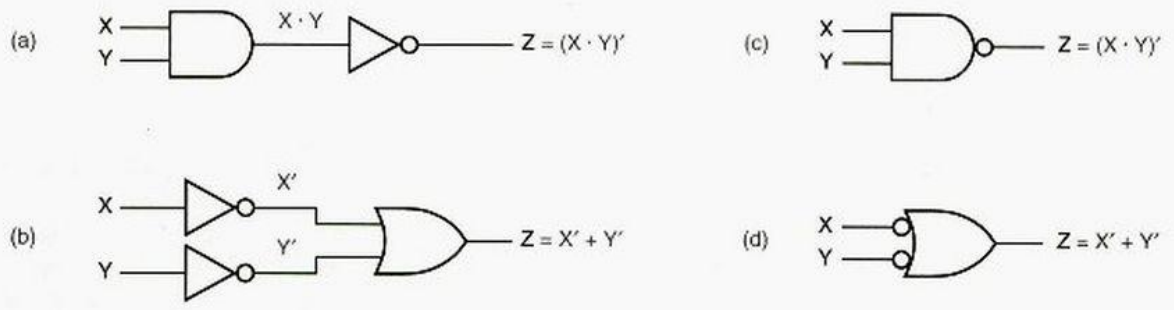
(T12)	$X + X + \cdots + X = X$	(Potencia idéntica generalizada)
(T12′)	$X \cdot X \cdot \ \cdots \ \cdot X = X$	
(T13)	$(X_1 \cdot X_2 \cdot \ \cdots \ \cdot X_n)' = X_1' + X_2' + \cdots + X_n'$	(Teoremas de DeMorgan)
(T13′)	$(X_1 + X_2 + \cdots + X_n)' = X_1' \cdot X_2' \cdot \ \cdots \ \cdot X_n'$	
(T14)	$[F(X_1, X_2, \ldots, X_n, +, \cdot)]' = F(X_1', X_2', \ldots, X_n', \cdot, +)$	(Teorema de DeMorgan generalizado)
(T15)	$F(X_1, X_2, \ldots, X_n) = X_1 \cdot F(1, X_2, \ldots, X_n) + X_1' \cdot F(0, X_2, \ldots, X_n)$	(Teoremas de expansión de Shannon)
(T15′)	$F(X_1, X_2, \ldots, X_n) = [X_1 + F(0, X_2, \ldots, X_n)] \cdot [X_1' + F(1, X_2, \ldots, X_n)]$	

una compuerta OR de *n* entradas cuyas entradas están complementadas. Es decir, los circuitos de la figura 4-3(a) y 4-3(b) son equivalentes.

En la sección 3.3.4 mostramos cómo construir una compuerta NAND CMOS. La salida de una compuerta NAND para cualquier conjunto de entradas es el complemento de una salida de compuerta AND para las mismas entradas, de modo que una compuerta NAND tiene el símbolo lógico de la figura 4-3(c). Sin embargo, el circuito NAND CMOS no está diseñado como una compuerta AND seguida de un inversor con transistor (compuerta NOT); es sólo una colección de transistores que da la casualidad que realizan la función AND-NOT. De hecho, el teorema T13 nos dice que el símbolo lógico en (d) denota la misma función lógica (las burbujas sobre las entradas de la compuerta OR indican inversión lógica). Es decir, la compuerta NAND se considera como si realizara una función NOT-OR.

Al observar las entradas y la salida de una compuerta NAND, es imposible determinar si la compuerta se ha construido internamente como una compuerta AND seguida de un inversor, o como inversores seguidos de una compuerta OR, o como una realización CMOS directa, *porque todos los circuitos NAND realizan la misma función lógica.* Aunque la selección del símbolo no tiene la menor influencia sobre la funcionalidad del circuito, en la sección 5.1 demostraremos que la selección apropiada hace la función del circuito mucho más fácil de comprender.

Figura 4-3 Circuitos equivalentes de acuerdo con el teorema de DeMorgan T13: a) AND-NOT; b) NOT-OR; c) símbolo lógico para una compuerta NAND; d) símbolo equivalente para una compuerta NAND.



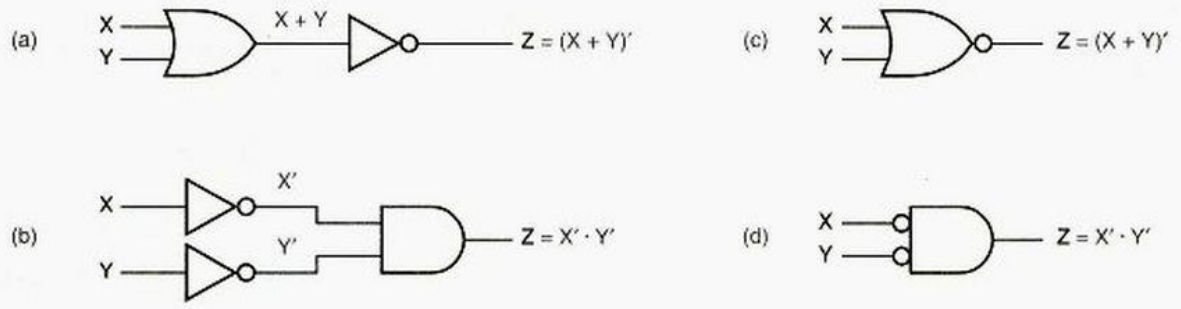


Figura 4-4 Circuitos equivalentes de acuerdo con el teorema de DeMorgan T13': a) OR-NOT; b) NOT-AND; c) símbolo lógico para una compuerta NOR; d) símbolo equivalente para una compuerta NOR.

Una equivalencia simbólica semejante se infiere del teorema T13′. Como se muestra en la figura 4-4, una compuerta NOR puede llevarse a cabo como una compuerta OR seguida de un inversor, o como inversores seguidos de una compuerta AND.

teorema generalizado de DeMorgan

complemento de una expresión lógica

Los teoremas T13 y T13′ son casos especiales de un *teorema generalizado de DeMorgan*, T14, que se aplica a una expresión lógica arbitraria F. Por definición, el *complemento de una expresión lógica*, que se representa como (F)′, es una expresión cuyo valor es el opuesto de las F's para toda combinación posible de entrada. El teorema T14 es muy importante debido a que nos proporciona una manera de manipular y simplificar el complemento de una expresión.

El teorema T14 establece que dada una expresión lógica de n variables, se obtiene su complemento al intercambiar + y · y complementar todas las variables. Por ejemplo, supongamos que tenemos

$$\begin{aligned} F(W, X, Y, Z) &= (W' \cdot X) + (X \cdot Y) + (W \cdot (X' + Z')) \\ &= ((W)' \cdot X) + (X \cdot Y) + (W \cdot ((X)' + (Z)')) \end{aligned}$$

En la segunda línea hemos encerrado las variables complementadas entre paréntesis para recordarle que la ′ es un operador, no parte del nombre de la variable. Al aplicar el teorema T14, obtenemos lo siguiente:

$$[F(W, X, Y, Z)]' = ((W')' + X') \cdot (X' + Y') \cdot (W' + ((X')' \cdot (Z')'))$$

Aplicando del teorema T4, esto se simplifica a

$$[F(W, X, Y, Z)]' = (W + X') \cdot (X' + Y') \cdot (W' + (X \cdot Z))$$

En general, utilizamos el teorema T14 para complementar una expresión entre paréntesis mediante el intercambio de + y · , complementando todas las variables que están sin complementar y dejando sin complementar todas aquéllas que se encuentren complementadas.

El teorema generalizado de DeMorgan T14 se demuestra mostrando que todas las funciones lógicas se escriben ya sea como una suma o como un producto de subfunciones, y se aplican luego de manera recursiva T13 y T13′. Sin embargo, una demostración

mucho más satisfactoria e instructiva se basa en el principio de dualidad, que se explica a continuación.

4.1.5 Dualidad

Hemos establecido todos los axiomas del álgebra de conmutación en parejas. La versión "prima" de cada axioma (por ejemplo, A5′) se obtiene de la versión sin prima (por ejemplo, A5) al intercambiar sencillamente 0 y 1 y, si se encuentran presentes, · y +. Como resultado, podemos establecer el siguiente *metateorema*, un teorema acerca de teoremas: *metateorema*

Principio de dualidad Cualquier teorema o identidad en el álgebra de conmutación continúa siendo verdadero si tanto 0 y 1 como · y + son intercambiados en todas partes.

El metateorema es verdadero debido a que los duales de todos los axiomas son verdaderos, de modo que los duales de todos los teoremas del álgebra de conmutación se demuestran utilizando los duales de los axiomas.

Después de todo, ¿qué es un nombre o un símbolo en todo caso? Si el software empleado para la tipografía de este libro hubiera tenido un error, uno que intercambiara $0 \leftrightarrow 1$ y $\cdot \leftrightarrow +$ a lo largo de este capítulo, usted todavía hubiera aprendido exactamente la misma álgebra de conmutación; solamente la nomenclatura habría sido un poco extraña, utilizando palabras como "producto" para describir una operación que utiliza el símbolo "+".

La dualidad es importante puesto que duplica la utilidad de todo lo que se aprende acerca del álgebra de conmutación y la manipulación de las funciones de conmutación. De una manera más práctica, desde el punto de vista de un estudiante, ¡reduce a la mitad la cantidad de lo que usted tiene que aprender! Por ejemplo, una vez que usted ha aprendido cómo sintetizar los circuitos lógicos AND-OR de dos etapas, a partir de expresiones de suma de productos, automáticamente sabrá una técnica dual para sintetizar circuitos OR-AND a partir de expresiones de producto de sumas.

Hay únicamente una convención en el álgebra de conmutación donde no se trata idénticamente · y +, de modo que la dualidad no necesariamente se mantiene verdadera: ¿puede usted averiguar de qué se trata antes de leer la respuesta abajo? Considere la siguiente instrucción del teorema T9 y su claramente absurdo "dual":

$$X + X \cdot Y = X \quad \text{(teorema T9)}$$

$$X \cdot X + Y = X \quad \text{(después de aplicar el principio de dualidad)}$$

$$X + Y = X \quad \text{(después de aplicar el teorema T3′)}$$

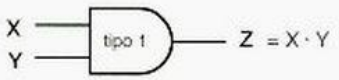
Obviamente la última línea anterior es falsa: ¿dónde cometimos la equivocación? El problema se encuentra en la precedencia de operadores. Fuimos capaces de escribir el lado izquierdo de la primera línea sin paréntesis debido a nuestra convención de que · tiene precedencia. Sin embargo, una vez que se aplica el principio de dualidad, se debe dar precedencia a + en su lugar, o escrito la segunda línea como $X \cdot (X + Y) = X$. La mejor manera de evitar problemas como éste es colocar explícitamente todos los paréntesis de una expresión antes de tomar su dual.

Definamos formalmente el *dual de una expresión lógica*. Si $F(X_1, X_2, \ldots, X_n, +, \cdot, ')$ es una expresión lógica con paréntesis explícitos que involucra las variables $X_1, X_2, \ldots, X_n$ *dual de una expresión lógica*

(a)

X	Y	Z
BAJO	BAJO	BAJO
BAJO	ALTO	BAJO
ALTO	BAJO	BAJO
ALTO	ALTO	ALTO

(b)



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

(c)

X	Y	Z
1	1	1
1	0	1
0	1	1
0	0	0

Figura 4-5 Una compuerta lógica "tipo 1": a) tabla de funciones eléctricas; b) tabla de funciones lógicas y símbolo con lógica positiva; c) tabla de funciones lógicas y símbolo con lógica negativa.

y los operadores +, ·, y ′, entonces el dual de F, escrito como F^D, es la misma expresión con + y · intercambiados:

$$F^D(X_1, X_2, \ldots, X_n, +, \cdot, ') = F(X_1, X_2, \ldots, X_n, +, \cdot, ')$$

Naturalmente, usted ya sabía esto, pero escribimos la definición de esta manera precisamente para destacar la similitud entre la dualidad y el teorema T14 generalizado de DeMorgan, el cual será ahora nuevamente establecido como sigue:

$$[F(X_1, X_2, \ldots, X_n)]' = F^D(X_1', X_2', \ldots, X_n')$$

Examinemos este enunciado desde el punto de vista de una red física.

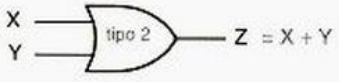
La figura 4-5(a) muestra la tabla de funciones eléctricas de un elemento lógico al que designaremos compuerta "tipo 1". Bajo la convención de lógica positiva (BAJO = 0 y ALTO = 1), esto es una compuerta AND, pero bajo la convención de lógica negativa (BAJO = 1 y ALTO = 0), es una compuerta OR, como se ilustra en (b) y (c). También podemos imaginar una compuerta "tipo 2", la cual se ilustra en la figura 4-6, que es una OR de lógica positiva o una AND de lógica negativa. Para compuertas con más de dos entradas se pueden desarrollar tablas semejantes.

Figura 4-6 Una compuerta lógica "tipo 2": a) tabla de funciones eléctricas; b) tabla de funciones lógicas y símbolo con lógica positiva; c) tabla de funciones lógicas y símbolo con lógica negativa.

(a)

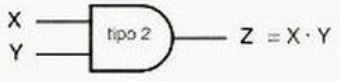
X	Y	Z
BAJO	BAJO	BAJO
BAJO	ALTO	ALTO
ALTO	BAJO	ALTO
ALTO	ALTO	ALTO

(b)



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

(c)



X	Y	Z
1	1	1
1	0	0
0	1	0
0	0	0

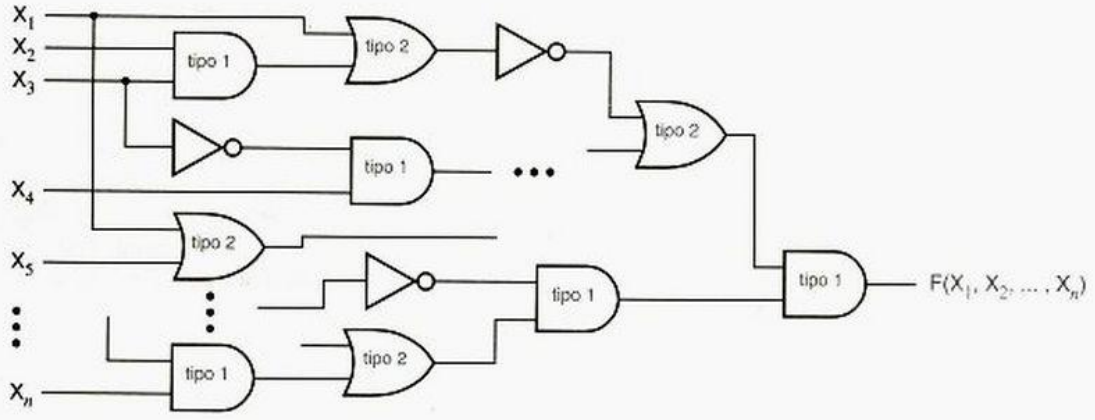


Figura 4-7 Circuito para una función lógica utilizando inversores y compuertas tipo 1 y tipo 2 bajo una convención de lógica positiva.

Supongamos que estamos dando una expresión de lógica arbitraria, $F(X_1, X_2, \ldots, X_n)$. Siguiendo la convención de lógica positiva, construimos el circuito correspondiente a esta expresión utilizando inversores para operaciones NOT, compuertas tipo 1 para AND y compuertas tipo 2 para OR, como se ilustra en la figura 4-7. Ahora supongamos que, sin cambiar este circuito, sencillamente modifiquemos la convención de lógica de positiva a negativa. Entonces deberíamos volver a dibujar el circuito como se muestra en la figura 4-8. Claramente, para toda combinación posible de voltajes de entrada (ALTO y BAJO), el circuito todavía produce el mismo voltaje de salida. Sin embargo, desde el punto de vista del álgebra de conmutación, el valor de salida (0 o 1) es el opuesto del que se tenía bajo la convención de lógica positiva. Del mismo modo, cada valor de entrada es el opuesto del que se tenía. Por lo tanto, para cada combinación posible de entrada para el circuito de la figura 4-7, la salida es el opuesto de la producida por la combinación de entrada opuesta aplicada al circuito en la figura 4-8:

$$F(X_1, X_2, \ldots, X_n) = [F^D(X_1', X_2', \ldots, X_n')]'$$

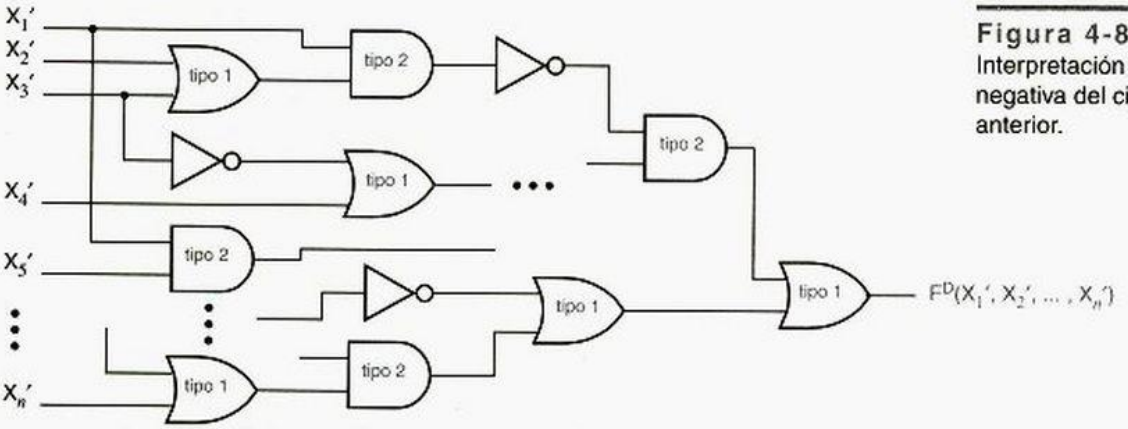


Figura 4-8
Interpretación en lógica negativa del circuito anterior.

Al complementar ambos lados, obtenemos el teorema generalizado de DeMorgan:

$$[F(X_1, X_2, ..., X_n)]' = F^D(X_1', X_2', ..., X_n')$$

¡Sorprendente!

LA DUALIDAD ES BUENA PARA ESTUDIANTES Y AUTORES

Usted ha visto que la dualidad es el fundamento para el teorema generalizado de DeMorgan. Yendo más allá, reducirá a la mitad el número de métodos que usted deberá aprender para manipular y simplificar funciones lógicas. ¡También reducirá a la mitad la cantidad de material original que yo tendría que escribir en estas secciones!

4.1.6 Representaciones estándar de funciones lógicas

Antes de trasladarnos al análisis y síntesis de las funciones lógicas combinacionales, presentaremos algo de nomenclatura y notación necesarias.

tabla de verdad

La representación más básica de una función lógica es la *tabla de verdad*. Similar al método de demostración por inducción perfecta en filosofía, esta representación por fuerza bruta simplemente enumera la salida del circuito para toda combinación posible de entrada. Tradicionalmente, las combinaciones de entrada se acomodan en renglones en orden ascendente de conteo binario, y los correspondientes valores de salida se escriben en una columna enseguida de los renglones. La estructura general de una tabla de verdad de 3 variables se muestra a continuación en la tabla 4-4.

Los renglones están numerados del 0 al 7, correspondientes a las combinaciones de entrada binaria, pero esta numeración no es una parte esencial de la tabla de verdad. La tabla de verdad para una función lógica de 3 variables se ilustra en la tabla 4-5. Cada patrón distinto de ceros y unos en la columna de salida produce una función lógica diferente; existen 2^8 patrones de esta clase. De este modo, la función lógica en la tabla 4-5 es una de 2^8 diferentes funciones lógicas de tres variables.

La tabla de verdad para una función lógica de n variables tiene 2^n renglones. Obviamente, las tablas de verdad son prácticas de escribir solamente para funciones lógicas con un número pequeño de variables, digamos, 10 para estudiantes y de 4 a 5 para cualquier otra persona.

Tabla 4-4
Estructura general de tabla de verdad para una función lógica de 3 variables, F(X, Y, Z).

Renglón	X	Y	Z	F
0	0	0	0	F(0,0,0)
1	0	0	1	F(0,0,1)
2	0	1	0	F(0,1,0)
3	0	1	1	F(0,1,1)
4	1	0	0	F(1,0,0)
5	1	0	1	F(1,0,1)
6	1	1	0	F(1,1,0)
7	1	1	1	F(1,1,1)

Renglón	X	Y	Z	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Tabla 4-5
Tabla de verdad para una función lógica particular de 3 variables, F(X, Y, Z).

La información contenida en una tabla de verdad también es posible expresarla algebraicamente. Para ello, necesitamos primero algunas definiciones:

- Una *literal* es una variable o el complemento de una variable. Ejemplos: X, Y, X′, Y′. *literal*
- Un *término de producto* es una literal simple o un producto lógico de dos o más literales. Ejemplos: Z', $W \cdot X \cdot Y$, $X \cdot Y' \cdot Z$, $W' \cdot Y' \cdot Z$. *término de producto*
- Una *expresión de suma de productos* es una suma lógica de términos de producto. Ejemplo: $Z' + W \cdot X \cdot Y + X \cdot Y' \cdot Z + W' \cdot Y' \cdot Z$. *expresión de suma de productos*
- Un *término de suma* es una literal simple o una suma lógica de dos o más literales. Ejemplos: Z', $W + X + Y$, $X + Y' + Z$, $W' + Y' + Z$. *término de suma*
- Una *expresión de producto de sumas* es un producto lógico de términos de suma. Ejemplo: $Z' \cdot (W + X + Y) \cdot (X + Y' + Z) \cdot (W' + Y' + Z)$. *expresión de producto de sumas*
- Un *término normal* es un término de producto o suma en el cual una variable aparece una sola vez. Un término no normal siempre se simplifica a una constante o un término normal utilizando uno de los teoremas T3, T3′, T5 o T5′. Ejemplos de términos no normales: $W \cdot X \cdot X \cdot Y'$, $W + W + X' + Y$, $X \cdot X' \cdot Y$. Ejemplos de términos normales: $W \cdot X \cdot Y'$, $W + X' + Y$. *término normal*
- Un *minitérmino* de n variables es un término de producto normal con n literales. Hay 2^n términos de producto de esta clase. Ejemplos de minitérminos de 4 variables: $W' \cdot X' \cdot Y' \cdot Z'$, $W \cdot X' \cdot Y' \cdot Z$, $W' \cdot X' \cdot Y \cdot Z'$. *minitérmino*
- Un *maxitérmino* de n variables es un término de suma normal con n literales. Hay 2^n términos de suma de esta clase. Ejemplos de maxitérminos de 4 variables: $W' + X' + Y' + Z'$, $W + X' + Y' + Z$, $W' + X' + Y + Z'$. *maxitérmino*

Existe una estrecha correspondencia entre la tabla de verdad y los minitérminos y maxitérminos. Un minitérmino puede también ser definido como un término de producto que es 1 en exactamente un renglón de la tabla de verdad. De manera semejante, un maxitérmino puede también ser definido como un término de suma que es 0 en exactamente un renglón de la tabla de verdad. La tabla 4-6 muestra esta correspondencia para una tabla de verdad de 3 variables.

Tabla 4-6
Minitérminos y maxitérminos para una función lógica de 3 variables, F(X, Y, Z).

Renglón	X	Y	Z	F	*Minitérmino*	*Maxitérmino*
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

número minitérmino
minitérmino i

Un *minitérmino* de n variables se representa por medio de un entero de n bits, el *número de minitérmino*. Utilizaremos el nombre *minitérmino i* para indicar el minitérmino correspondiente al renglón i de la tabla de verdad. En el minitérmino i, una variable particular aparece complementada si el bit correspondiente en la representación binaria de i es 0; de otro modo, no está complementada. Por ejemplo, el renglón 5 tiene una representación binaria 101 y el minitérmino correspondiente es $X \cdot Y' \cdot Z$. Como usted espera, la correspondencia para maxitérminos es precisamente la opuesta: en el *maxitérmino i*, una variable está complementada si el bit correspondiente en la representación binaria de i es 1. De este modo, el maxitérmino 5 (101) es $X' + Y + Z'$. Note que todo esto tiene sentido solamente si conocemos el número de variables en la tabla de verdad, tres en los ejemplos.

maxitérmino i

suma canónica

Con base en la correspondencia entre la tabla de verdad y los minitérminos, es posible crear fácilmente una representación algebraica de una función lógica a partir de su tabla de verdad. La *suma canónica* de una función lógica es una suma de los minitérminos correspondiente a los renglones de la tabla de verdad (combinaciones de entrada) para los cuales la función produce una salida 1. Por ejemplo, la suma canónica para la función lógica en la tabla 4-5 de la página 207 es

$$\begin{aligned} F &= \Sigma_{X,Y,Z}(0, 3, 4, 6, 7) \\ &= X' \cdot Y' \cdot Z' + X' \cdot Y \cdot Z + X \cdot Y' \cdot Z' + X \cdot Y \cdot Z' + X \cdot Y \cdot Z \end{aligned}$$

lista de minitérminos

conjunto de iniciación

Aquí, la notación $\Sigma_{X,Y,Z}(0, 3, 4, 6, 7)$ es una *lista de minitérminos* y quiere decir "la suma de los minitérminos 0, 3, 4, 6 y 7 con variables X, Y y Z". La lista de minitérminos también se conoce como el *conjunto de iniciación (on-set)* para la función lógica. Podemos visualizar que cada minitérmino "enciende" exactamente la salida para una combinación de entrada. Cualquier función lógica puede escribirse como una suma canónica.

producto canónico

El *producto canónico* de una función lógica es un producto de los maxitérminos correspondientes a combinaciones de entrada para las cuales la función produce una salida 0. Por ejemplo, el producto canónico para la función lógica en la tabla 4-5 es

$$\begin{aligned} F &= \Pi_{X,Y,Z}(1, 2, 5) \\ &= (X + Y + Z') \cdot (X + Y' + Z) \cdot (X' + Y + Z') \end{aligned}$$

Aquí, la notación $\Pi_{X,Y,Z}(1,2,5)$ es una *lista de maxitérminos* y quiere decir "el producto de *maxitérminos* 1, 2 y 5 con variables X, Y y Z". La lista de maxitérminos también se conoce como el *conjunto de neutralización (off-set)* para la función lógica. Usted visualiza que cada maxitérmino "apaga" la salida para exactamente una combinación de entrada. Cualquier función lógica puede escribirse como un producto canónico.

lista de maxitérminos
conjunto de neutralización

Es fácil hacer la conversión entre una lista de minitérminos y una lista de maxitérminos. Para una función de n variables, los posibles números minitérmino y maxitérmino se encuentran en el conjunto $\{0, 1, ..., 2^n - 1\}$; una lista de minitérminos o maxitérminos contiene un subconjunto de estos números. Para hacer el intercambio entre tipos de lista, tome el complemento del conjunto, por ejemplo,

$$\Sigma_{A,B,C}(0,1,2,3) = \Pi_{A,B,C}(4,5,6,7)$$

$$\Sigma_{X,Y}(1) = \Pi_{X,Y}(0,2,3)$$

$$\Sigma_{W,X,Y,Z}(0,1,2,3,5,7,11,13) = \Pi_{W,X,Y,Z}(4,6,8,9,10,12,14,15)$$

Ahora hemos aprendido cinco posibles representaciones para una función lógica combinacional:

1. Una tabla de verdad.
2. Una suma algebraica de minitérminos, la suma canónica.
3. Una lista de minitérminos que utilizan la notación Σ.
4. Un producto algebraico de maxitérminos, el producto canónico.
5. Una lista de maxitérminos que utilizan la notación Π.

Cada una de estas representaciones especifica exactamente la misma información; dada cualquiera de ellas, las otras cuatro se obtienen utilizando un proceso mecánico sencillo.

4.2 Análisis de circuitos combinacionales

Analicemos un circuito lógico combinacional mediante la obtención de una descripción formal de su función lógica. Una vez que tenemos una descripción de la función lógica, se pueden realizar otras operaciones:

- Determinar el comportamiento del circuito para varias combinaciones de entrada.
- Manipular una descripción algebraica para sugerir diferentes estructuras de circuito para la función lógica.
- Transformar una descripción algebraica en una forma estándar que corresponde a una estructura de circuito disponible. Por ejemplo, una expresión de suma de productos corresponde directamente a la estructura de circuito que se utiliza en dispositivos de lógica programable (PLD, por sus siglas en inglés).
- Se puede utilizar una descripción algebraica del comportamiento funcional del circuito en el análisis de un sistema más grande que incluya al circuito.

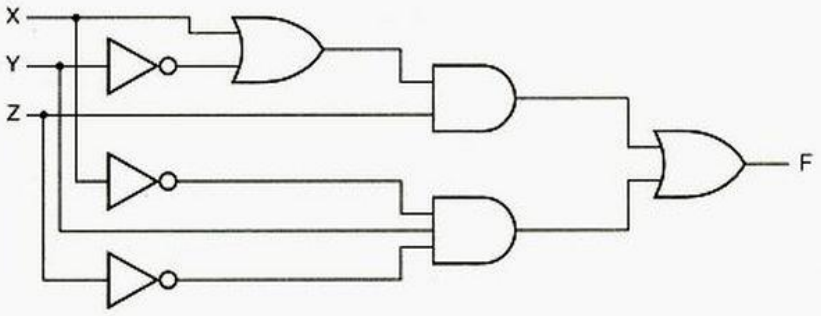
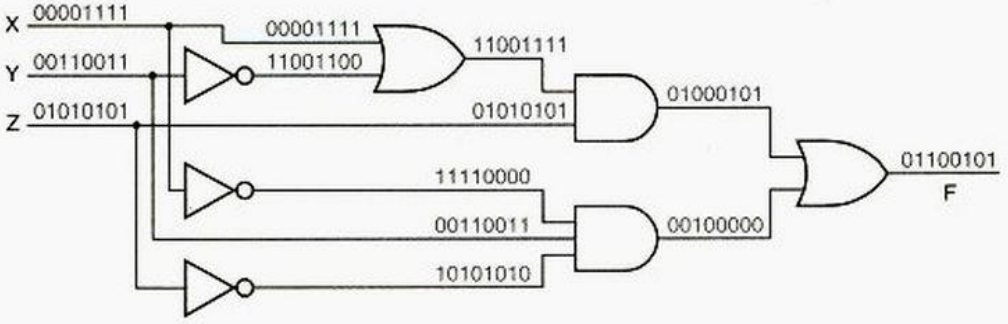


Figura 4-9
Un circuito lógico de tres entradas y una salida.

Una vez que se tiene el diagrama lógico de un circuito combinacional, tal como el de la figura 4-9, existen varias maneras de obtener una descripción formal de la función del circuito. La descripción funcional más primitiva es la tabla de verdad.

Si solamente utilizamos los axiomas básicos del álgebra de conmutación, obtenemos la tabla de verdad de un circuito de n entradas trabajando a través de todas las 2^n combinaciones de entrada. Para cada combinación de entrada, determinamos todas las salidas de compuerta producidas por esa entrada, propagando la información desde las entradas del circuito hasta las salidas del mismo. La figura 4-10 aplica esta "exhaustiva" técnica a nuestro circuito de ejemplo. En cada línea de señal del circuito hay una serie de ocho valores lógicos, los valores que están presentes en esa línea cuando las entradas XYZ del circuito son 000, 001, ..., 111. Escribimos la tabla de verdad al transcribir la serie de salida de la compuerta OR final, como se muestra en la tabla 4-7. Una vez que tenemos la tabla de

Figura 4-10 Salidas de compuertas creadas por todas las combinaciones de entradas.



UNA MANERA MENOS EXHAUSTIVA DE HACER LAS COSAS

Usted puede obtener fácilmente los resultados de la figura 4-10 con herramientas de diseño lógico típicas que incluyan un simulador lógico. En primer lugar, se dibuja el esquema. Luego, se aplican las salidas de un contador binario de 3 bits a las entradas X, Y y Z. (La mayoría de los simuladores tienen tales salidas de contador integradas justamente para esta clase de ejercicio.) El contador pasa repetidas veces a través de las ocho posibles combinaciones de entrada, en el mismo orden que hemos mostrado en la figura. El simulador nos permite graficar los valores de señal resultantes en cualquier punto del esquema, incluyendo los puntos intermedios así como también la salida.

Renglón	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Tabla 4-7
Tabla de verdad para el circuito lógico de la figura 4-9.

verdad para el circuito, podemos obtener directamente una expresión lógica (la suma o producto canónicos), si así lo deseamos.

El número de combinaciones de entrada de un circuito lógico crece de manera exponencial con el número de entradas, de modo que el enfoque exhaustivo puede rápidamente llegar a ser agotador. En su lugar, normalmente utilizamos un enfoque algebraico cuya complejidad es más linealmente proporcional al tamaño del circuito. El método es sencillo; construimos una expresión lógica con paréntesis correspondiente a los operadores lógicos y a la estructura del circuito. Comenzamos por las entradas del circuito y propagamos las expresiones a través de las compuertas hasta la salida. Utilizando los teoremas del álgebra de conmutación, simplificamos las expresiones a medida que las encontramos, o diferimos todas las manipulaciones algebraicas hasta obtener una expresión de salida.

La figura 4-11 aplica la técnica algebraica a nuestro circuito de ejemplo. La función de salida está dada a la salida de la compuerta OR final:

$$F = ((X+Y') \cdot Z) + (X' \cdot Y \cdot Z')$$

Ningún teorema del álgebra de conmutación se utilizó para obtener esta expresión. Sin embargo, empleamos los teoremas para transformar esta expresión en otra forma. Por ejemplo, puede obtenerse una suma de productos al "expandir":

$$F = X \cdot Z + Y' \cdot Z + X' \cdot Y \cdot Z'$$

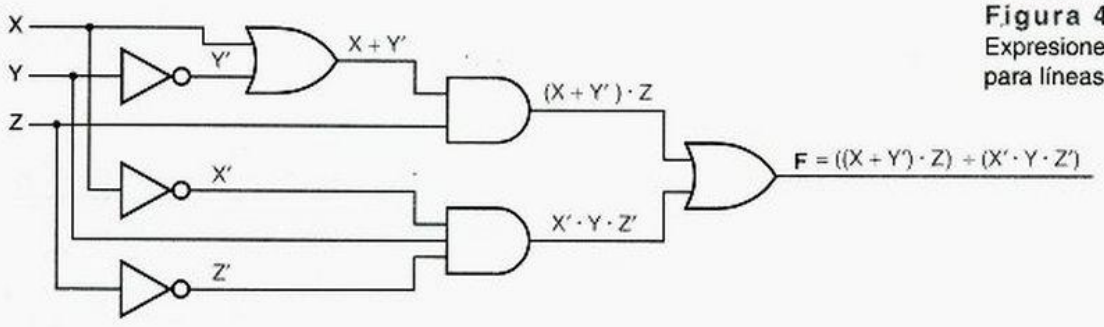


Figura 4-11
Expresiones lógicas para líneas de señal.

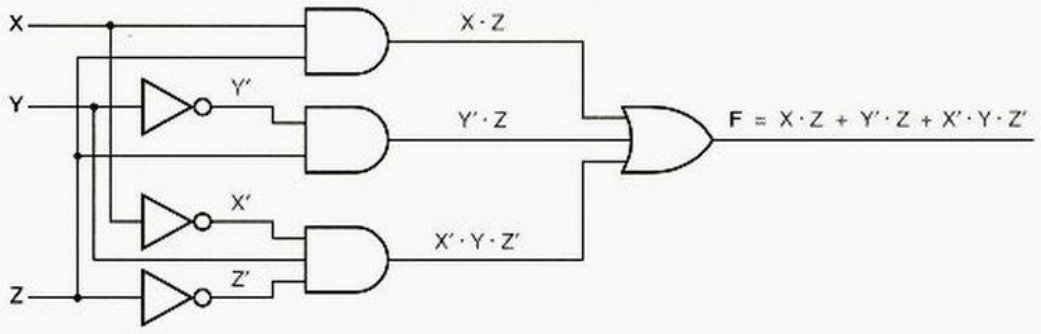


Figura 4-12 Circuito AND-OR de dos niveles.

La nueva expresión corresponde a un circuito diferente que realiza la misma función lógica, como se ilustra en la figura 4-12.

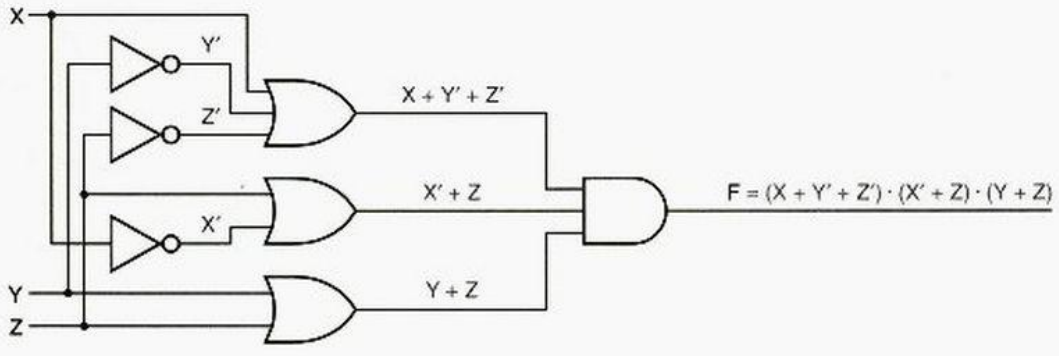
De manera similar, se "expande la suma" de la expresión original para obtener un producto de sumas:

$$
\begin{aligned}
F &= ((X + Y') \cdot Z) + (X' \cdot Y \cdot Z') \\
&= (X + Y' + X') \cdot (X + Y' + Y) \cdot (X + Y' + Z') \cdot (Z + X') \cdot (Z + Y) \cdot (Z + Z') \\
&= 1 \cdot 1 \cdot (X + Y' + Z') \cdot (X' + Z) \cdot (Y + Z) \cdot 1 \\
&= (X + Y' + Z') \cdot (X' + Z) \cdot (Y + Z)
\end{aligned}
$$

El circuito lógico correspondiente se muestra en la figura 4-13.

Nuestro siguiente ejemplo de análisis algebraico utiliza un circuito con compuertas NAND y NOR, como se muestra en la figura 4-14. Este análisis es un poco más complicado que en el ejemplo anterior, porque cada compuerta produce una subexpresión complementada, no sólo una suma o producto simples. Sin embargo, la expresión de salida se simplifica mediante la aplicación repetida del teorema generalizado de DeMorgan:

Figura 4-13 Circuito OR-AND de dos niveles.



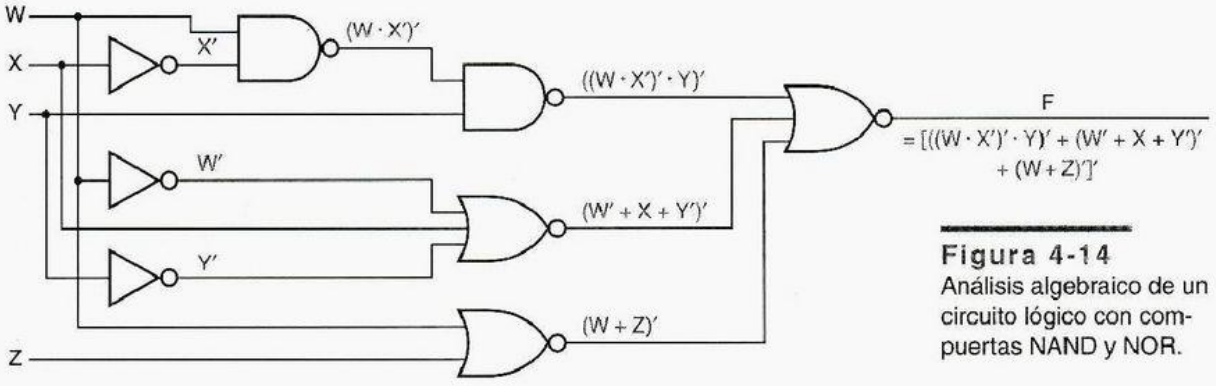


Figura 4-14
Análisis algebraico de un circuito lógico con compuertas NAND y NOR.

$$F = [((W \cdot X')' \cdot Y)' + (W' + X + Y')' + (W + Z)']'$$
$$= ((W \cdot X')' \cdot Y) \cdot (W' + X + Y') \cdot (W + Z)$$
$$= ((W' + X) \cdot Y) \cdot (W' + X + Y') \cdot (W + Z)$$

Con bastante frecuencia, el teorema de DeMorgan se aplica *gráficamente* para simplificar el análisis algebraico. Recuerde de las figuras 4-3 y 4-4 que las compuertas NAND y NOR tienen cada una dos símbolos equivalentes. Al volver a trazar prudentemente la figura 4-14, hacemos posible la cancelación de algunos de los inversores durante el análisis al utilizar el teorema T4 [$(X')' = X$], como se muestra en la figura 4-15. Esta manipulación nos conduce de manera directa a una expresión de salida simplificada:

$$F = ((W' + X) \cdot Y) \cdot (W' + X + Y') \cdot (W + Z)$$

Las figuras 4-14 y 4-15 fueron solamente dos maneras diferentes de dibujar el mismo circuito lógico físico. Sin embargo, cuando simplificamos una expresión lógica empleando los teoremas del álgebra de conmutación, obtenemos una expresión correspondiente a un circuito físico diferente. Por ejemplo, la expresión simplificada anterior corresponde al circuito de la figura 4-16, el cual es físicamente diferente del que aparece en las dos figuras anteriores. Además, podríamos distribuir el producto y la suma de

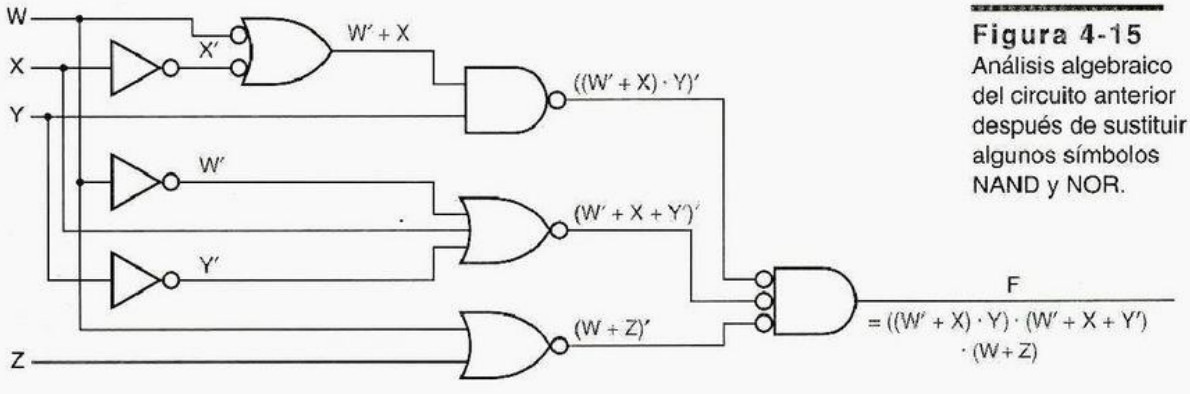


Figura 4-15
Análisis algebraico del circuito anterior después de sustituir algunos símbolos NAND y NOR.

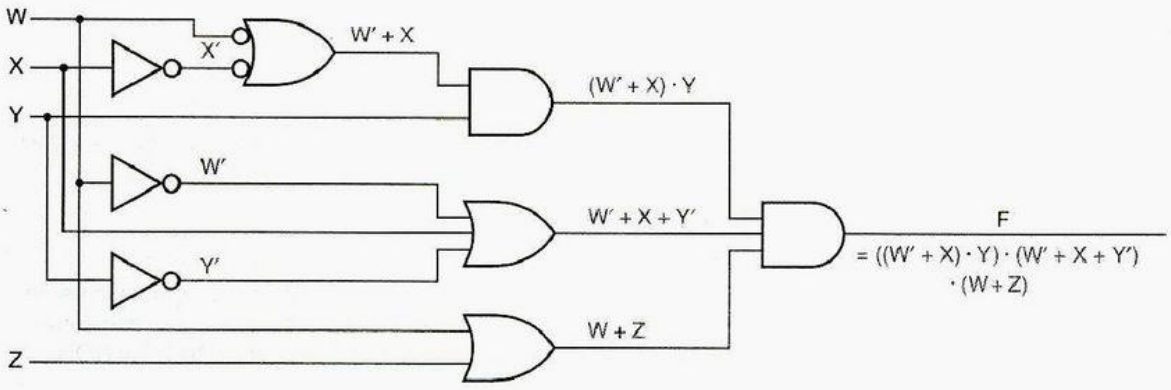
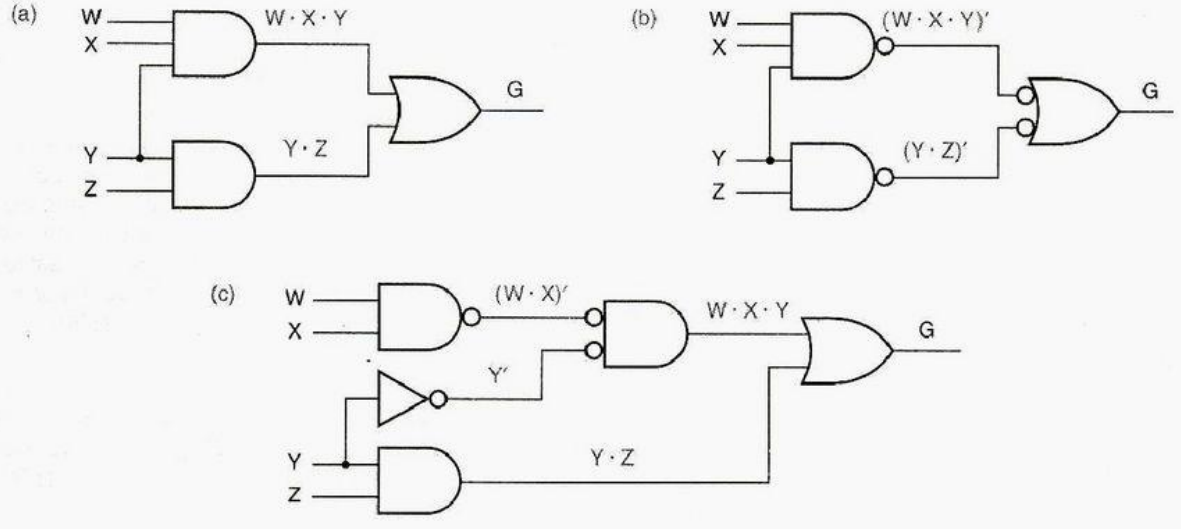


Figura 4-16 Un circuito diferente para la misma función lógica.

la expresión para obtener expresiones de suma de productos y producto de sumas, que corresponderían a dos circuitos adicionales, que son físicamente diferentes pero realizan la misma función lógica.

Aunque anteriormente utilizamos expresiones lógicas para obtener información acerca de la estructura física de un circuito, esto es algo que no siempre se realiza. Por ejemplo, empleamos las expresiones $G(W, X, Y, Z) = W \cdot X \cdot Y + Y \cdot Z$ para describir cualquiera de los circuitos en la figura 4-17. Normalmente, la única forma segura para determinar una estructura de circuito es examinar su diagrama esquemático. Sin embargo, para ciertas clases restringidas de circuitos, la información estructural se infiere de las expresiones lógicas. Por ejemplo, el circuito en (a) se describe sin hacer referencia al dibujo como "un circuito AND-OR de dos niveles para $W \cdot X \cdot Y \cdot + Y \cdot Z$", mientras que el circuito en (b) se describe como "un circuito NAND-NAND de dos niveles para $W \cdot X \cdot Y + Y \cdot Z$".

Figura 4-17 Los tres circuitos para $G(W, X, Y, Z) = W \cdot X \cdot Y + Y \cdot Z$: a) AND-OR de dos niveles; b) NAND-NAND de dos niveles; c) ad hoc.



4.3 Síntesis de circuitos combinacionales

4.3.1 Diseños y descripciones de circuito

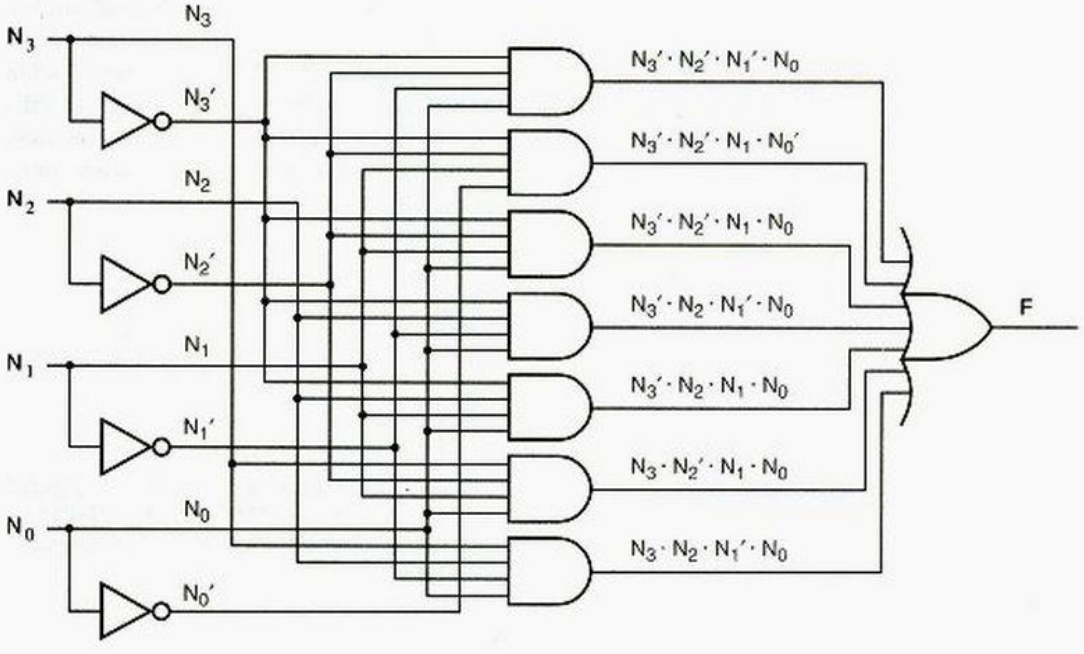
¿Cuál es el punto de partida para diseñar circuitos lógicos combinacionales? Por lo regular, tenemos una descripción en palabras de un problema o desarrollamos una nosotros mismos. Ocasionalmente, la descripción es una lista de combinaciones de entrada para las cuales una señal debería encenderse o apagarse, el equivalente verbal de una tabla de verdad o de la notación Σ o Π presentadas con anterioridad. Por ejemplo, puede ser la descripción de un detector de números primos de 4 bits, "dada una combinación de entrada de 4 bits, $N = N_3N_2N_1N_0$, esta función produce una salida 1 para N = 1, 2, 3, 5, 7, 11, 13 y 0 de otro modo". Una función lógica descrita de esta manera se diseña directamente a partir de la suma canónica de expresión del producto. Para el detector de números primos tenemos

$$\begin{aligned} F &= \Sigma_{N_3N_2N_1N_0}(1, 2, 3, 5, 7, 11, 13) \\ &= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2' \cdot N_1' \cdot N_0 \\ &\quad + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + N_3 \cdot N_2' \cdot N_1 \cdot N_0 + N_3 \cdot N_2 \cdot N_1' \cdot N_0 \end{aligned}$$

El circuito correspondiente se ilustra en la figura 4-18.

Con más frecuencia, describimos una función lógica aplicando las conjunciones en idioma inglés tales como "AND, OR y NOT". Por ejemplo, podemos describir un circuito de alarma diciendo "la salida ALARMA es 1 si la entrada PÁNICO es 1, O (OR) si la entrada HABILITAR es 1, la salida SALIR es 0 Y (AND) la casa NO (NOT) es

Figura 4-18 Diseño de suma canónica para detector de números primos de 4 bits.



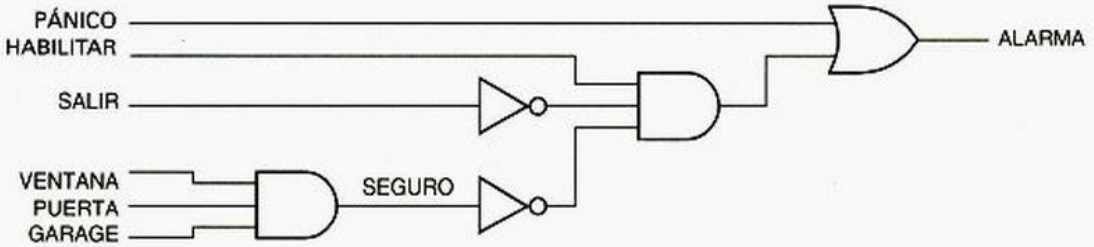


Figura 4-19 Circuito de alarma derivado de la expresión lógica.

segura; la casa es segura si todas las entradas VENTANA, PUERTA y GARAGE son 1". Una descripción de esta clase se traduce directamente en expresiones algebraicas:

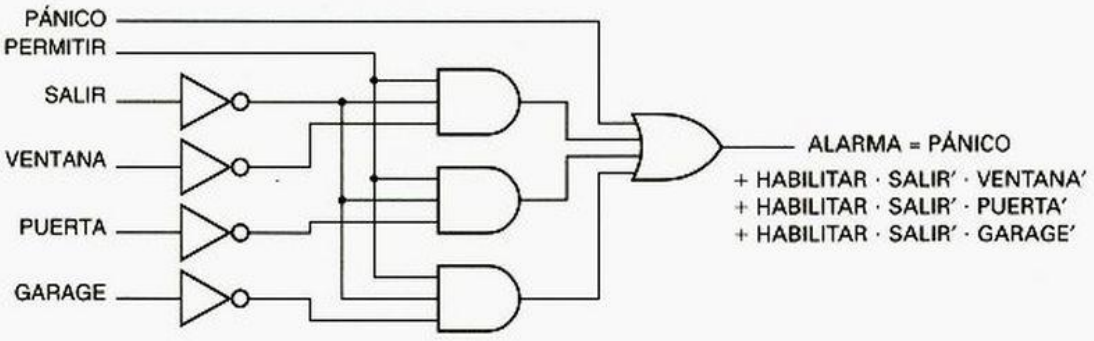
$$\text{ALARMA} = \text{PÁNICO} + \text{HABILITAR} \cdot \text{SALIDA}' \cdot \text{ASEGURAR}'$$
$$\text{ASEGURAR} = \text{VENTANA} \cdot \text{PUERTA} \cdot \text{GARAGE}$$
$$\text{ALARMA} = \text{PÁNICO} + \text{HABILITAR} \cdot \text{SALIDA}' \cdot (\text{VENTANA} \cdot \text{PUERTA} \cdot \text{GARAGE})'$$

Note que utilizamos el mismo método en el álgebra de conmutación que en el álgebra ordinaria para formular una expresión complicada: definimos una variable auxiliar SEGURO para simplificar la primera ecuación, desarrollamos una expresión para SEGURO y empleamos la sustitución para obtener la expresión final. Dibujamos fácilmente un circuito utilizando compuertas AND, OR y NOT que realizan la expresión final, como se ilustra en la figura 4-19. Un circuito *realiza* ["hace realidad"] una expresión si su función de salida es igual a esa expresión, y el circuito se conoce como una *realización* de la función.

realizar
realización

Una vez que tenemos una expresión, cualquier expresión, para una función lógica, hacemos otras cosas aparte de construir un circuito directamente de la expresión. Podemos manipular la expresión para obtener diferentes circuitos. Por ejemplo, la expresión anterior ALARMA se expande para obtener el circuito de suma de productos en la figura 4-20. Por otra parte, si el número de variables no es demasiado grande, construimos la tabla de verdad de la expresión y aplicamos cualquiera de los métodos de síntesis que se aplican a las tablas de verdad, incluyendo los métodos de suma o producto canónico descritos con anterioridad y los métodos de minimización que se describen más adelante.

Figura 4-20 Versión para suma de productos del circuito de alarma.



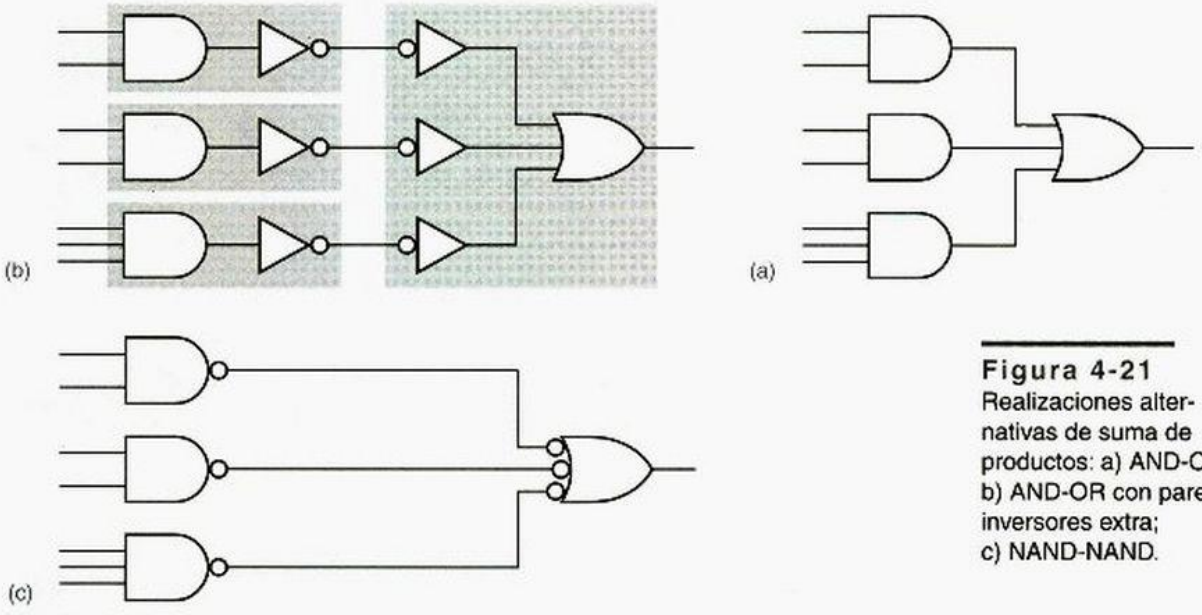


Figura 4-21
Realizaciones alternativas de suma de productos: a) AND-OR; b) AND-OR con pares inversores extra; c) NAND-NAND.

En general, es más fácil describir un circuito con palabras. Para esto se utilizan conjunciones o conectores lógicos y se escriben las correspondientes expresiones lógicas, esto es mejor que escribir una tabla de verdad completa, especialmente si la cantidad de variables es grande. Sin embargo, en ocasiones tenemos que trabajar con descripciones orales nada precisas de las funciones lógicas, por ejemplo, "La salida ERROR debería ser 1 si las entradas ENGRANEARRIBA, ENGRANEABAJO y VERIFICAENGRANE son inconsistentes". En esta situación, el enfoque de la tabla de verdad es mejor porque nos permite determinar la salida que se requiere para cada combinación de entrada, basándonos en nuestro conocimiento y comprensión del entorno del problema (por ejemplo, los frenos no se aplican a menos que haya reducción "abajo" del engranaje).

4.3.2 Manipulaciones de circuito

Los métodos de diseño que describimos hasta ahora hacen uso de compuertas AND, OR y NOT. Podemos desear utilizar compuertas NAND y NOR, también: son más rápidas que las ANDs y ORs en la mayoría de las tecnologías. Sin embargo, la mayor parte de las personas no desarrollan instrucciones lógicas en términos de conectores NAND y NOR. Es decir, usted probablemente no diría "no te daré una cita si eres no decente o no adinerado y también si eres no inteligente o no amable". Sería más natural decir "Te daré una cita si eres decente y adinerado, o si eres inteligente y amable". Así, dada una expresión lógica "natural", necesitamos maneras de traducirla en otras formas.

Es posible traducir cualquier expresión lógica en una expresión de suma de productos equivalente, al simplemente expandirla. Como se muestra en la figura 4-21(a), una expresión de esta clase se realiza directamente con compuertas AND y OR. Los inversores requeridos para las entradas complementadas no se ilustran.

Como se muestra en la figura 4-21(b), podemos insertar un par de inversores entre cada salida de compuerta AND y la correspondiente entrada de compuerta OR en un

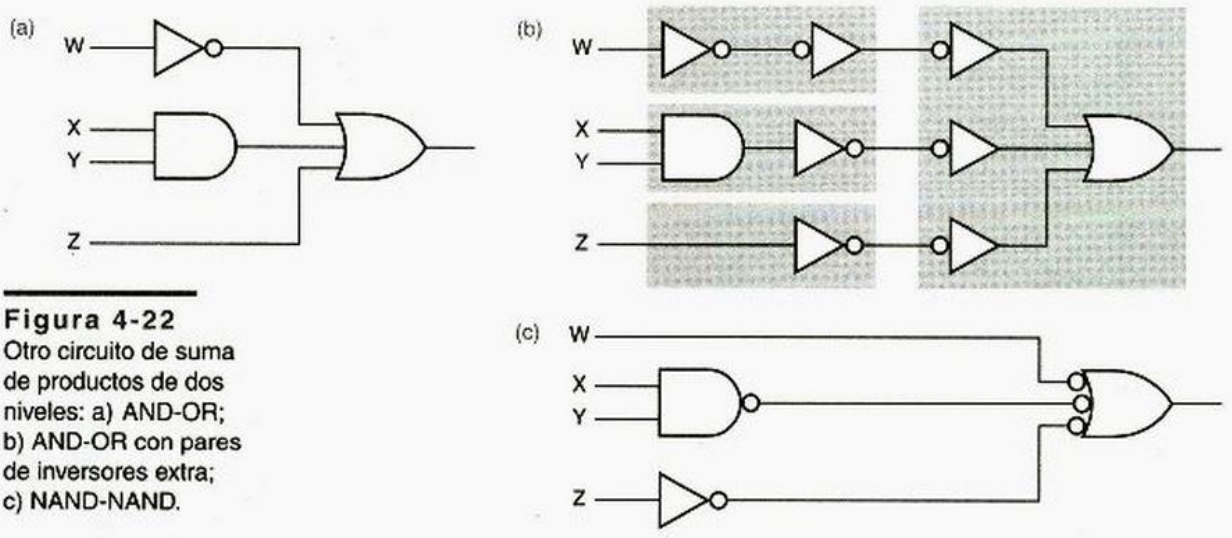


Figura 4-22
Otro circuito de suma de productos de dos niveles: a) AND-OR; b) AND-OR con pares de inversores extra; c) NAND-NAND.

circuito AND-OR de dos niveles. De acuerdo con el teorema T4, estos inversores no tienen efecto sobre la función de salida del circuito. De hecho, dibujamos el segundo inversor de cada par con su burbuja de inversión en su entrada para proporcionar un recordatorio gráfico de que los inversores se cancelan. Sin embargo, si estos inversores son absorbidos dentro de las compuertas AND y OR, acabamos con compuertas AND-NOT en el primer nivel y con una compuerta NOT-OR en el segundo nivel. Éstos son precisamente dos símbolos diferentes para el mismo tipo de compuerta: una compuerta NAND. De este modo, un *circuito* AND-OR de dos niveles se convierte en un *circuito* NAND-NAND de dos niveles sencillamente sustituyendo las compuertas.

circuito AND-OR
circuito NAND-NAND

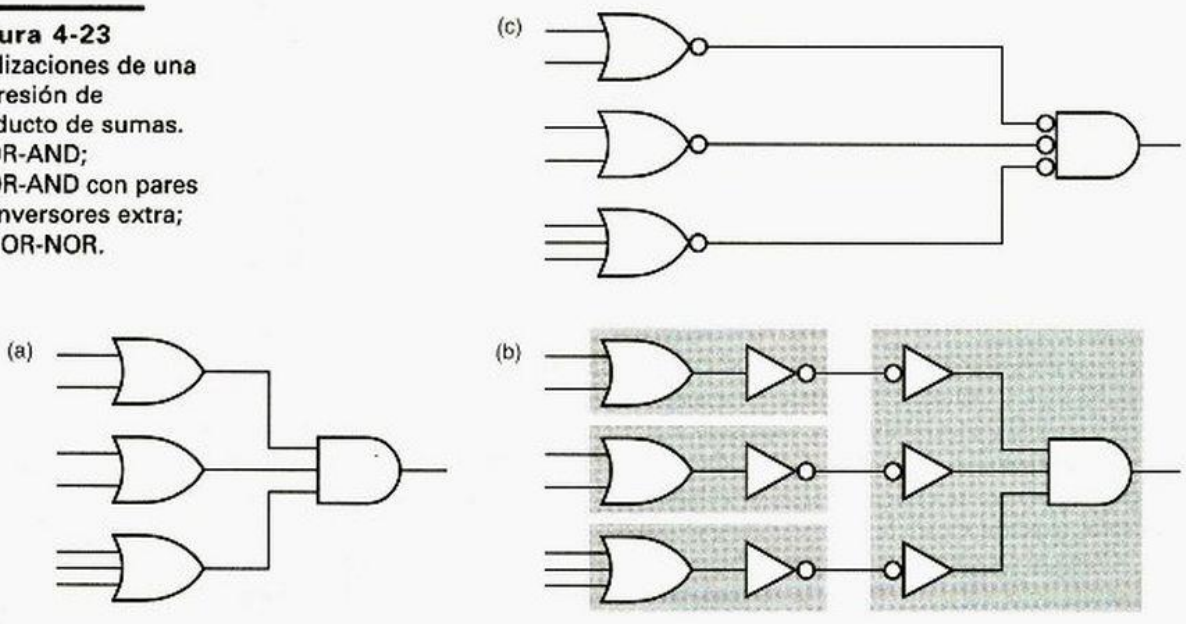


Figura 4-23
Realizaciones de una expresión de producto de sumas. a) OR-AND; b) OR-AND con pares de inversores extra; c) NOR-NOR.

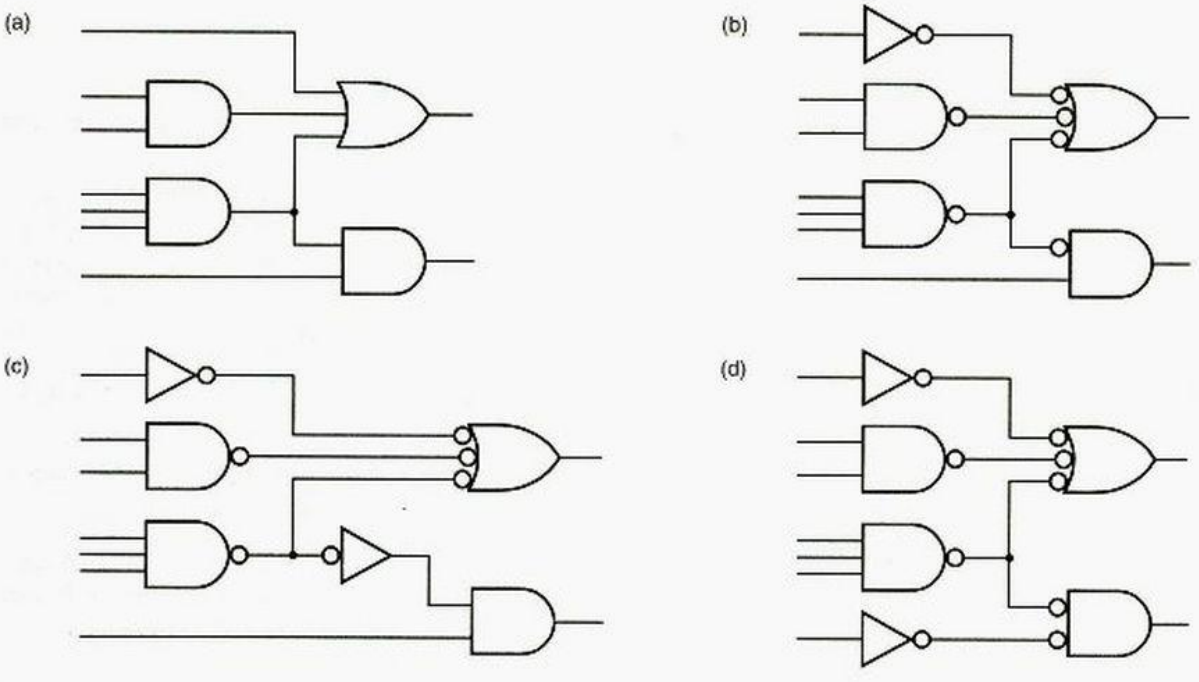
Si cualquier término de producto en la expresión de suma de producto, contiene apenas una sola literal, entonces ganaremos o perderemos inversores en la transformación de AND-OR a NAND-NAND. Por ejemplo, la figura 4-22 es un ejemplo donde un inversor en la entrada W ya no es necesario, pero se debe agregar un inversor a la entrada Z.

Hemos mostrado que cualquier expresión de suma de productos puede realizarse en cualesquiera de dos formas: como un circuito AND-OR o como un circuito NAND-NAND. El dual de este enunciado también es cierto: cualquier expresión de producto de sumas puede realizarse como un *circuito* OR-AND o como un *circuito* NOR-NOR. La figura 4-23 ilustra un ejemplo. Cualquier expresión lógica se traduce en una expresión equivalente de producto de sumas mediante la expansión de la suma, y de ahí que tenga realizaciones de ambos circuitos OR-AND y NOR-NOR.

circuito OR-AND
circuito NOR-NOR

La misma clase de manipulaciones se aplica a circuitos lógicos arbitrarios. Por ejemplo, la figura 4-24(a) ilustra un circuito construido a partir de compuertas AND y OR. Después de agregar pares de inversores, obtenemos el circuito en (b). Sin embargo, una de las compuertas, una compuerta AND de 2 entradas con una entrada simple invertida, no es de un tipo estándar. Podemos utilizar un inversor discreto como se muestra en (c) para obtener un circuito que emplee únicamente tipos estándar de compuertas: NAND, AND e inversores. En realidad, una mejor manera de utilizar el inversor se muestra en (d); se elimina un nivel de retardo de la compuerta y la compuerta inferior se convierte en una NOR en vez de AND. En la mayoría de las tecnologías lógicas, las compuertas inversoras como NAND y NOR son más rápidas que las compuertas no inversoras tales como AND y OR.

Figura 4-24 Manipulaciones de símbolo lógico: a) circuito original; b) transformación con una compuerta no estándar; c) inversor utilizado para eliminar la compuerta no estándar; d) colocación del inversor preferido.



¿POR QUÉ SE DEBE MINIMIZAR?

La minimización es un paso importante tanto en el diseño ASIC como en el diseño basado en PLD. Las compuertas adicionales y las entradas de las compuertas requieren mayor área en un chip ASIC y por ello aumenta el costo. La cantidad de compuertas en un PLD es fijo, de modo que se puede pensar que las compuertas extra son gratis. Y lo son, hasta que se agotan y se tiene que actualizar el diseño a un PLD más grande, lento y costoso. Afortunadamente, la mayoría de las herramientas de software tanto para diseño ASIC como para PLD tienen un programa de minimización integrado. El propósito de las secciones 4.3.3 a la 4.3.8 es que usted experimente y aprenda cómo funciona la minimización.

4.3.3 Minimización del circuito combinacional

Con frecuencia no es económico realizar un circuito lógico directamente de la primera expresión lógica que aparezca en su cabeza. Las expresiones de suma y producto canónicos son especialmente costosas debido a que el número de posibles minitérminos y maxitérminos (y por tanto compuertas) crece exponencialmente con el número de variables. *Minimizamos* un circuito combinacional mediante la reducción del número y tamaño de las compuertas que son necesarias para construirlo.

minimizar

Los métodos tradicionales de minimización de circuitos combinacionales que estudiaremos tienen como punto de partida una tabla de verdad o, de manera equivalente, una lista de minitérminos o una lista de maxitérminos. Si damos una función lógica que no esté expresada en esta forma, entonces debemos convertirla a una forma apropiada antes de emplear estos métodos. Por ejemplo, si damos una expresión lógica arbitraria, entonces la evaluaremos para cada combinación de entrada a fin de construir la tabla de verdad. Los métodos de minimización reducen el costo de un circuito de dos niveles AND-OR, OR-AND, NAND-NAND o NOR-NOR de tres maneras:

1. Minimizando el número de compuertas de primer nivel.
2. Minimizando el número de entradas en cada compuerta de primer nivel.
3. Minimizando el número de entradas en la compuerta de segundo nivel. Esto es en realidad un efecto colateral de la primera reducción.

Sin embargo, los métodos de minimización no consideran el costo de los inversores de entrada; suponen que tanto las versiones verdaderas como las complementadas de todas las variables de entrada se encuentran disponibles. Mientras que éste no siempre es el caso en el diseño ASIC o al nivel de compuerta, es muy apropiado para el diseño basado en PLD: los PLD tienen versiones tanto verdaderas como complementadas de todas las variables de entrada disponibles, "gratis".

La mayoría de los métodos de minimización se basan en una generalización de los teoremas de combinación, T10 y T10′:

término de producto determinado · Y + término de producto determinado · Y′ = término de producto determinado

(término de suma dado + Y) · (término de suma dado + Y′) = término de suma dado

Es decir, si dos términos de producto o suma difieren solamente en el complemento o negación de una variable, los combinamos en un único término con una variable menos. De este modo ahorramos una compuerta y la compuerta restante tienen una entrada menos.

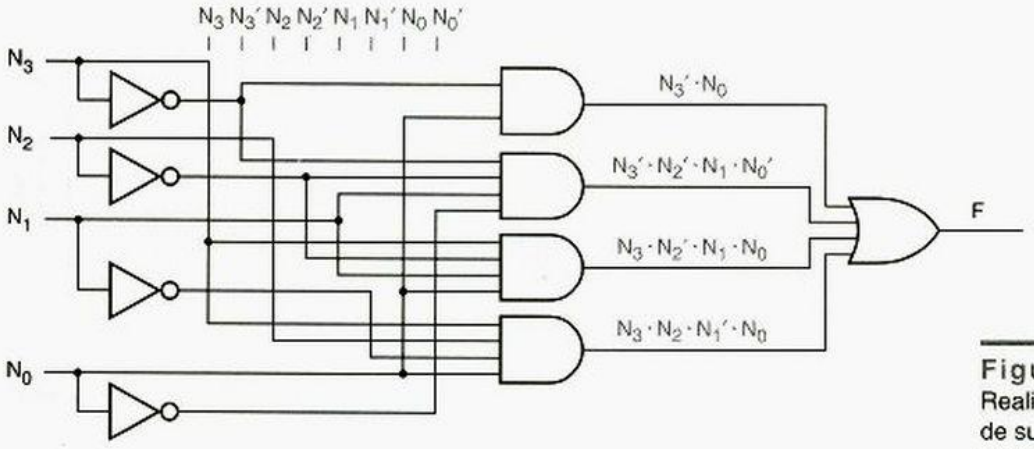


Figura 4-25
Realización simplificada de suma de productos para el detector de números primos de 4 bits.

Es posible aplicar este método algebraico repetidamente para combinar los minitérminos 1, 3, 5 y 7 del detector de números primos mostrado en la figura 4-18 de la página 215:

$$
\begin{aligned}
F &= \Sigma_{N_3,N_2,N_1,N_0}(1, 3, 5, 7, 2, 11, 13) \\
&= N_3' \cdot N_2' N_1' N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + \ldots \\
&= (N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0) + (N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0) + \ldots \\
&= N_3' N_2' \cdot N_0 + N_3' \cdot N_2 \cdot N_0 + \ldots \\
&= N_3' \cdot N_0 + \ldots
\end{aligned}
$$

El circuito resultante se ilustra en la figura 4-25; tiene tres compuertas menos y una de las compuertas restantes tiene dos entradas menos.

Si hubiéramos trabajado un poco más en la expresión anterior, habríamos ahorrado un par más de entradas de compuerta de primer nivel, aunque ninguna compuerta. Es difícil hallar términos que se combinen en una mezcolanza de símbolos algebraicos. En la subsección siguiente comenzaremos a explorar un método de minimización que es más adecuado para el consumo humano. Nuestro punto de partida será el equivalente gráfico de una tabla de verdad.

4.3.4 Mapas de Karnaugh

mapa de Karnaugh

Un *mapa de Karnaugh* es una representación gráfica de una tabla de verdad de la función lógica. La figura 4-26 muestra mapas de Karnaugh para funciones lógicas de 2, 3 y 4 variables. El mapa para una función lógica de n entradas es un arreglo con 2^n celdas, una por cada posible combinación de entrada o minitérmino.

Los renglones y columnas de un mapa de Karnaugh se etiquetan de tal modo que la combinación de entrada para cualquier celda se determina fácilmente de los encabezados de renglón y columna para esa celda. El pequeño número dentro de cada celda es el correspondiente número de minitérmino en la tabla de verdad, suponiendo que las entradas de la tabla de verdad se encuentran etiquetadas alfabéticamente de izquierda a derecha (por ejemplo, X, Y, Z) y los renglones estén numerados en orden de conteo binario, como

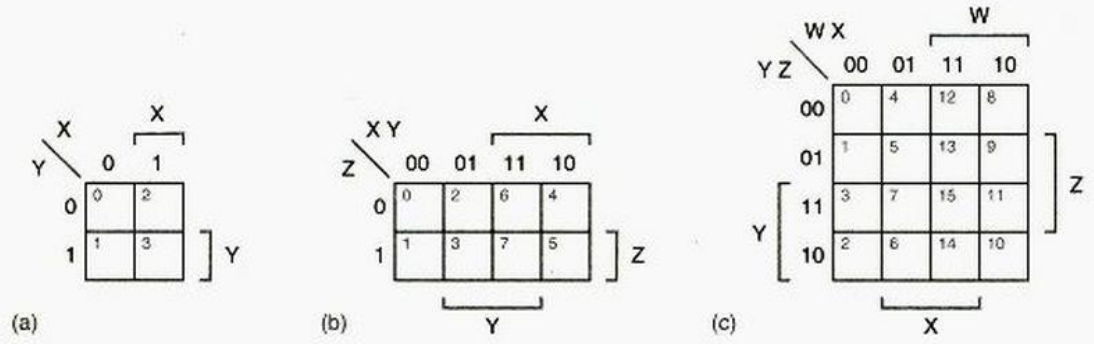


Figura 4-26 Mapas de Karnaugh: a) de 2 variables; b) de 3 variables; c) de 4 variables.

todos los ejemplos en este texto. Por ejemplo, la celda 13 en el mapa de 4 variables corresponde al renglón de la tabla de verdad en el que W X Y Z = 1101.

Cuando dibujamos el mapa de Karnaugh para una función dada, cada celda del mapa contiene la información del renglón numerado de la tabla de verdad de la función: un 0 si la función es 0 para esa combinación de entrada, o un 1 de otro modo.

En este texto utilizamos dos etiquetados redundantes para renglones y columnas del mapa. Por ejemplo, considere el mapa de 4 variables en la figura 4-26(c). Las columnas se etiquetan con las cuatro posibles combinaciones de W y X, W X = 00, 01, 11 y 10. De manera semejante, los renglones se etiquetan con las combinaciones Y Z. Estas etiquetas nos proporcionan toda la información que necesitamos. Sin embargo, también utilizamos corchetes (paréntesis cuadrados) para asociar cuatro regiones del mapa con las cuatro variables. Cada región entre corchetes es la parte del mapa en la que la variable indicada es 1. Obviamente, los corchetes transmiten la misma información que es dada por las etiquetas de renglón y columna.

Cuando dibujamos un mapa a mano, es mucho más fácil dibujar los corchetes que escribir todas las etiquetas. Sin embargo, conservamos las etiquetas en los mapas de Karnaugh del texto como una ayuda adicional para el aprendizaje. En cualquier caso, debe estar seguro de etiquetar los renglones y columnas en el orden adecuado para mantener la correspondencia entre las celdas del mapa y los números de renglón de la tabla de verdad mostrada en la figura 4-26.

Para representar una función lógica sobre un mapa de Karnaugh, simplemente copiamos los unos y los ceros de la tabla de verdad o equivalente a las correspondientes celdas del mapa. Las figuras 4-27(a) y (b) muestran la tabla de verdad y el mapa de Karnaugh para una función lógica que analizamos (¿destrozamos?) en la sección 4-2. Desde ahora, reduciremos el desorden en los mapas al copiar solamente los unos o los ceros, no ambos.

4.3.5 Minimización de sumas o productos

En este momento, usted debe estar sorprendido por la "extraña" ordenación de los números de renglón y columna en un mapa de Karnaugh. Existe una muy importante razón para esta ordenación: cada celda corresponde a una combinación de entrada que difiere de cada una de sus vecinas inmediatamente adyacentes en sólo una variable. Por ejemplo, las celdas 5 y 13 en el mapa de 4 variables difieren solamente en el valor de W. En los

(a)

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

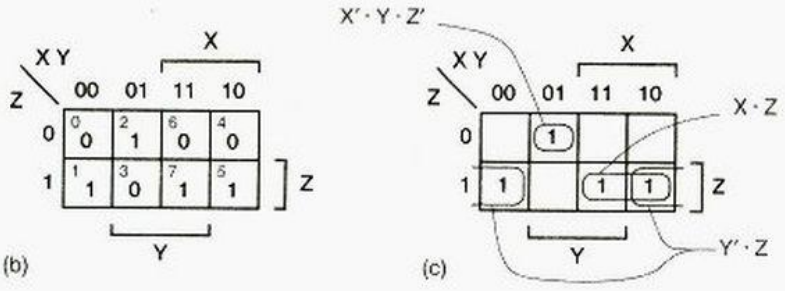


Figura 4-27 $F = \Sigma_{X,Y,Z}(1,2,5,7)$: a) tabla de verdad; b) mapa de Karnaugh; c) combinación de celdas 1 adyacentes.

mapas de 3 y 4 variables, las celdas correspondientes en los bordes izquierdo/derecho y superior/inferior son los vecinos menos obvios; por ejemplo, las celdas 12 y 14 en el mapa de 4 variables son adyacentes porque difieren solamente en el valor de Y.

Cada combinación de entrada con un "1" en la tabla de verdad corresponde a un minitérmino en la suma canónica de la función lógica. Puesto que los pares de celdas "1" adyacentes en el mapa de Karnaugh tienen minitérminos que difieren en sólo una variable, los pares de minitérminos se combinan en un solo término de producto utilizando la generalización del teorema T10, término · Y + término · Y′ = término. De este modo, podemos emplear un mapa de Karnaugh para simplificar la suma canónica de una función lógica.

Por ejemplo, considere las celdas 5 y 7 en la figura 4-27(b) y su contribución a la suma canónica para esta función:

$$\begin{aligned} F &= \ldots + X \cdot Y' \cdot Z + X \cdot Y \cdot Z \\ &= \ldots + (X \cdot Z) \cdot Y' + (X \cdot Z) \cdot Y \\ &= \ldots + X \cdot Z \end{aligned}$$

Recordando la adyacencia de los extremos, vemos que las celdas 1 y 5 en la figura 4-27(b) también son adyacentes y pueden combinarse:

$$\begin{aligned} F &= X' \cdot Y' \cdot Z + X \cdot Y' \cdot Z + \ldots \\ &= X' \cdot (Y' \cdot Z) + X \cdot (Y' \cdot Z) + \ldots \\ &= Y' \cdot Z + \ldots \end{aligned}$$

En general, una función lógica se simplifica al combinar pares de celdas 1 adyacentes (minitérminos) siempre que sea posible y escribir una suma de términos de producto que cubran todas las celdas 1. La figura 4-27(c) muestra el resultado de nuestra función lógica de ejemplo. Encerramos en un círculo un par de unos para indicar que los minitérminos correspondientes están combinados en un solo término de producto. El circuito correspondiente AND-OR se ilustra en la figura 4-28.

En muchas funciones lógicas el procedimiento de combinación de celdas se extiende a combinar más de dos celdas 1 en un solo término de producto. Por ejemplo,

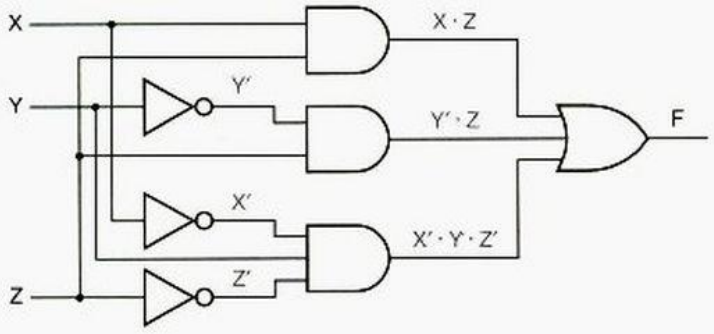


Figura 4-28
Circuito AND-OR minimizado.

considere la suma canónica para la función lógica $F = \Sigma_{X,Y,Z}(0, 1, 4, 5, 6)$. Utilizando las manipulaciones algebraicas de los ejemplos anteriores de manera iterativa para combinar cuatro de los cinco minitérminos, se tiene:

$$\begin{aligned} F &= X' \cdot Y' \cdot Z' + X' \cdot Y' \cdot Z + X \cdot Y' \cdot Z' + X \cdot Y' \cdot Z + X \cdot Y \cdot Z' \\ &= [(X' \cdot Y') \cdot Z' + (X' \cdot Y') \cdot Z] + [(X \cdot Y') \cdot Z' + (X \cdot Y') \cdot Z] + X \cdot Y \cdot Z' \\ &= X' \cdot Y' + X \cdot Y' + X \cdot Y \cdot Z' \\ &= [X' \cdot (Y') + X \cdot (Y')] + X \cdot Y \cdot Z' \\ &= Y' + X \cdot Y \cdot Z' \end{aligned}$$

En general, 2^i celdas 1 pueden ser combinadas para formar un término de producto conteniendo $n - i$ literales, donde n es el número de variables en la función.

Una regla matemática precisa determina cómo las celdas 1 se combinan y la forma del correspondiente término de producto:

- Un conjunto de 2^i celdas 1 se combina si existen i variables de la función lógica que tomen todas las 2^i posibles combinaciones dentro de ese conjunto, mientras que las restante $n - i$ variables tienen el mismo valor en todo ese conjunto. El término de producto correspondiente tiene $n - i$ literales, donde una variable está complementada si aparece como 0 en todas las celdas 1, y no está complementada si aparece como 1.

conjuntos rectangulares de unos

Gráficamente, esta regla quiere decir que podemos encerrar en un círculo conjuntos *rectangulares* de 2^n 1s, literalmente así como también extendiendo figurativamente la definición de rectangular para explicar la adyacencia en los extremos del mapa. Podemos determinar las literales de los correspondientes términos de producto directamente del mapa; para cada variable lo hacemos de la manera siguiente:

- Si un círculo cubre solamente las áreas del mapa donde la variable es 0, entonces la variable está complementada en el término de producto.
- Si un círculo cubre solamente las áreas del mapa donde la variable es 1, entonces la variable no está complementada en el término de producto.
- Si un círculo cubre tanto áreas del mapa donde la variable es 0 como donde la variable es 1, entonces la variable no aparece en el término de producto.

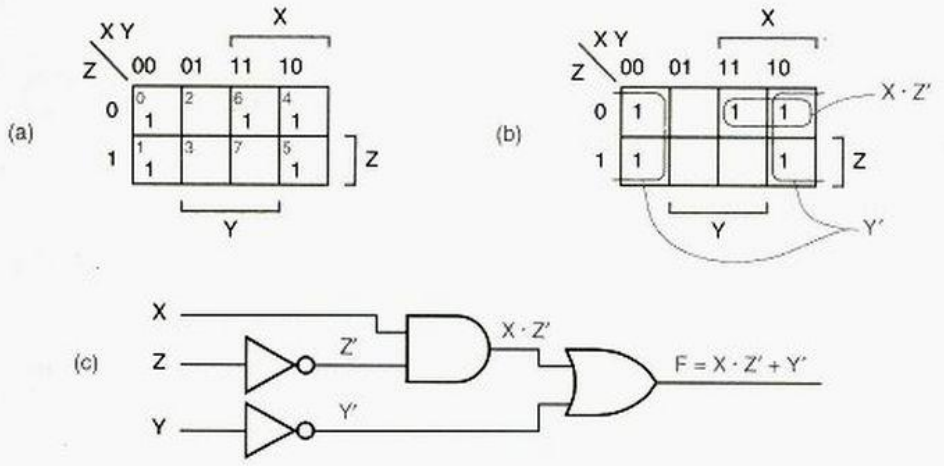


Figura 4-29
$F = \Sigma_{X,Y,Z}(0,1,4,5,6)$: a) mapa de Karnaugh inicial; b) mapa de Karnaugh con términos de producto en círculos; c) circuito AND/OR.

Una expresión de suma de productos para una función debe contener términos de producto (conjuntos encerrados en círculos de celdas 1) que cubran todos los unos y ninguno de los ceros en el mapa.

El mapa de Karnaugh para nuestro ejemplo más reciente, $F = \Sigma_{X,Y,Z}(0, 1, 4, 5, 6)$, se muestra en la figura 4-29(a) y (b). Hemos encerrado en un círculo un conjunto de cuatro unos, correspondientes al término de producto Y', y un conjunto de dos unos correspondientes al término de producto $X \cdot Z'$. Note que el segundo término de producto tiene sólo una literal menos que el término de producto correspondiente en nuestra solución algebraica ($X \cdot Y \cdot Z'$). Al encerrar en un círculo el conjunto más grande posible de unos conteniendo la celda 6, hemos hallado una realización menos costosa de la función lógica, puesto que una compuerta AND de 2 entradas debe costar menos que una de 3 entradas. El hecho de que dos diferentes términos de producto ahora cubran la misma celda 1 (4) no afecta la función lógica, ¡puesto que para la suma lógica 1 + 1 = 1, no 2! El correspondiente circuito AND/OR de dos niveles se muestra en (c).

Como otro ejemplo, el circuito detector de números primos que presentamos en la figura 4-18 de la página 215 puede ser minimizado como se ilustra en la figura 4-30.

En este punto, necesitamos algunas definiciones más para aclarar lo que estamos haciendo:

- Una *suma mínima* de una función lógica $F(X_1,\ldots,X_n)$ es una expresión de suma de productos para F de tal modo que ninguna expresión de suma de productos para F tiene menos términos de producto y cualquier expresión de suma de productos con el mismo número de términos de producto tiene al menos tantas literales. *suma mínima*

Es decir, la suma mínima tiene el menor número posible de términos de producto (compuertas de primer nivel y entradas de compuerta de segundo nivel) y, dentro de esta limitante, el menor número de literales posible (entradas de compuerta de primer nivel). Así, entre nuestros tres circuitos detectores de números primos, solamente el de la figura 4-30 en la siguiente página realiza una suma mínima.

La siguiente definición habla precisamente de lo que la palabra "implica" significa cuando se habla de funciones lógicas:

- Una función lógica $P(X_1,\ldots,X_n)$ *implica* una función lógica $F(X_1,\ldots,X_n)$ si para toda combinación de entrada tal que P = 1, entonces F = 1 también. *implicación*

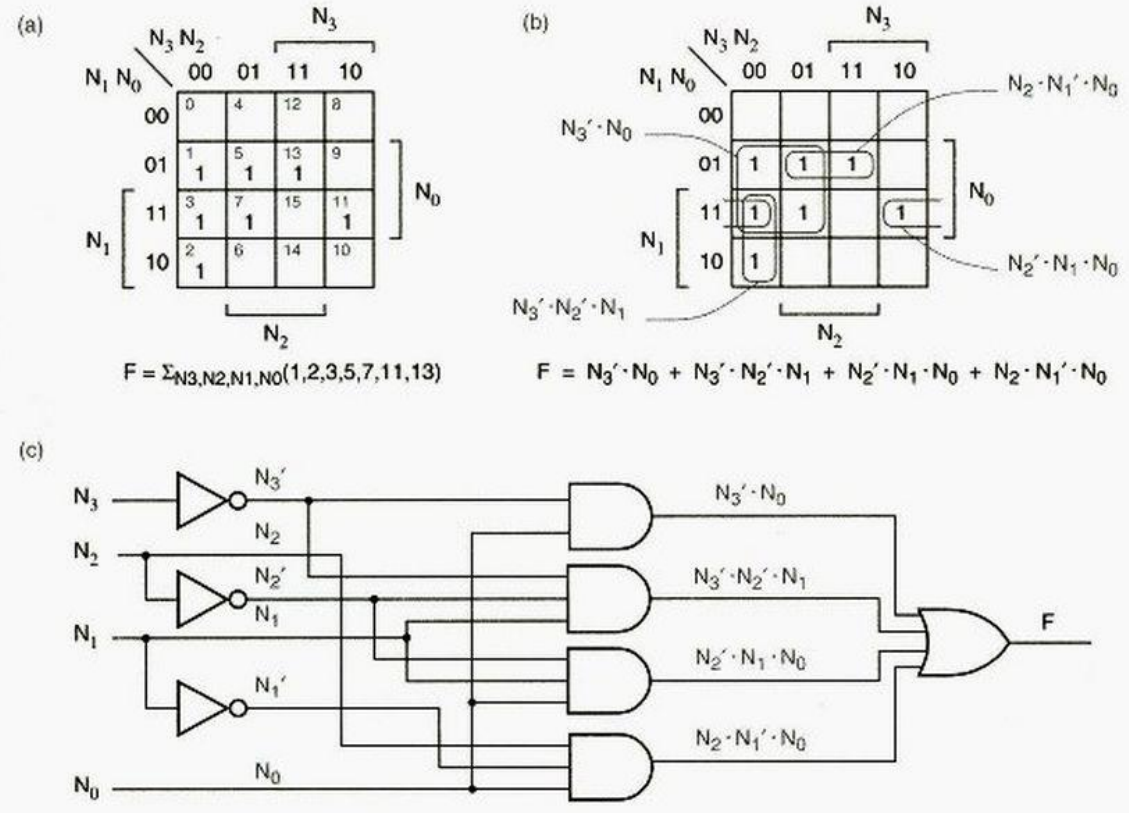


Figura 4-30 Detector de números primos: a) mapa de Karnaugh inicial; b) términos de producto en círculos; c) circuito minimizado.

incluye
cubre
implicante primo

Es decir, si P implica F, entonces F es 1 para toda combinación de entrada en que P es 1 y tal vez algo más. Podemos escribir la abreviatura $P \Rightarrow F$. También decimos que "F *incluye* P", o que "F *cubre a* P".

- El *implicante primo* de una función lógica $F(X_1,\ldots,X_n)$ es un término de producto normal $P(X_1,\ldots,X_n)$ que implica F, de tal modo que si cualquier variable es removida de P, entonces el término de producto resultante no implica F.

En términos de un mapa de Karnaugh, un implicante primo de F es un conjunto encerrado en círculo de celdas 1 que satisfacen nuestra regla de combinación, de tal suerte que si intentamos hacerlo más grande (cubriendo dos veces el número de celdas), cubre uno o más ceros.

Ahora viene la parte más importante, un teorema que limita cuánto trabajo debemos hacer para hallar una suma mínima para una función lógica:

Teorema del implicante primo Una suma mínima es una suma de implicantes primos.

Es decir, para encontrar una suma mínima, no necesitamos considerar ningún término de producto que no sea implicante primo. Este teorema es fácilmente demostrable por

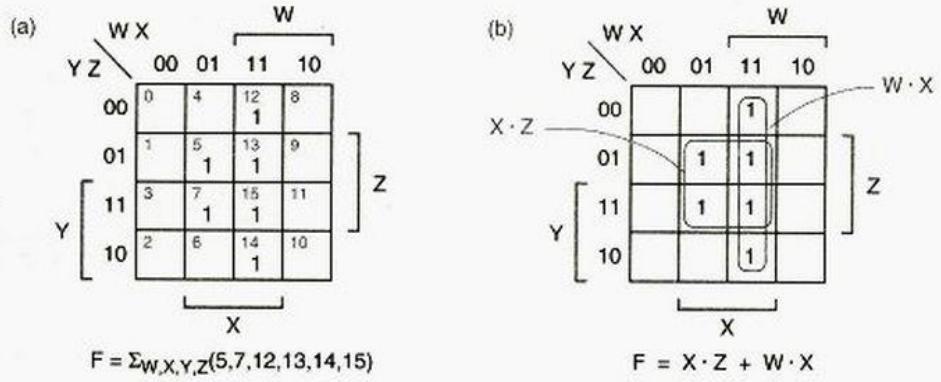


Figura 4-31 $F = \Sigma_{W,X,Y,Z}(5,7,12,13,14,15)$: (a) mapa de Karnaugh; b) implicantes primos.

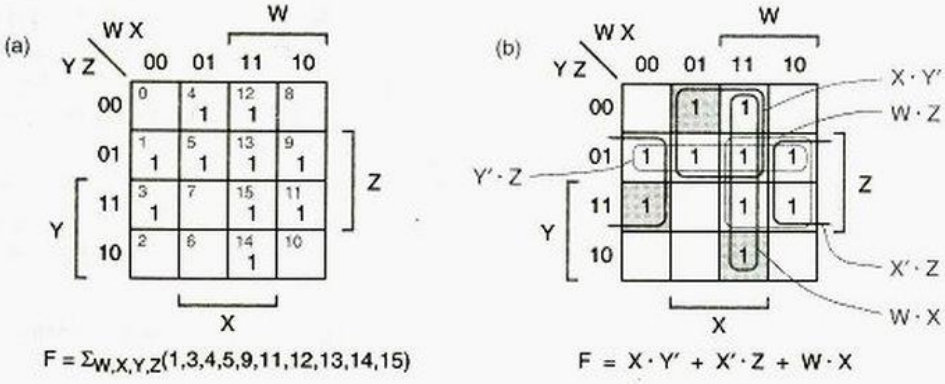
contradicción. Supongamos que un término de producto P en una suma "mínima" *no* es un implicante primo. Entonces de acuerdo a la definición de implicante primo, si P no es uno, es posible remover alguna literal de P para obtener un nuevo término de producto P* que todavía implique F. Si reemplazamos P con P* en la presumible suma "mínima", la suma resultante aún es igual a F pero tiene una literal menos. Por lo tanto, la presumible suma "mínima" no era mínima después de todo.

Otro ejemplo de minimización, esta vez una función de 4 variables, se muestra en la figura 4-31. Hay precisamente dos implicantes primos, y es bastante obvio que ambos deben ser incluidos en la suma mínima a fin de cubrir todas las celdas 1 en el mapa. No dibujamos el diagrama lógico para este ejemplo porque usted debe saber cómo hacerlo por usted mismo desde ahora.

La suma de todos los implicantes primos de una función lógica se denomina la *suma completa*. Aunque la suma completa es siempre una forma legítima de realizar una función lógica, no siempre es mínima. Por ejemplo, considere la función lógica mostrada en la figura 4-32. Tiene cinco implicantes primos, pero la suma mínima incluye solamente

suma completa

Figura 4-32 $F = \Sigma_{W,X,Y,Z}(1,3,4,5,9,11,12,13,14,15)$: (a) mapa de Karnaugh; b) implicantes primos y celdas 1 distinguidas.



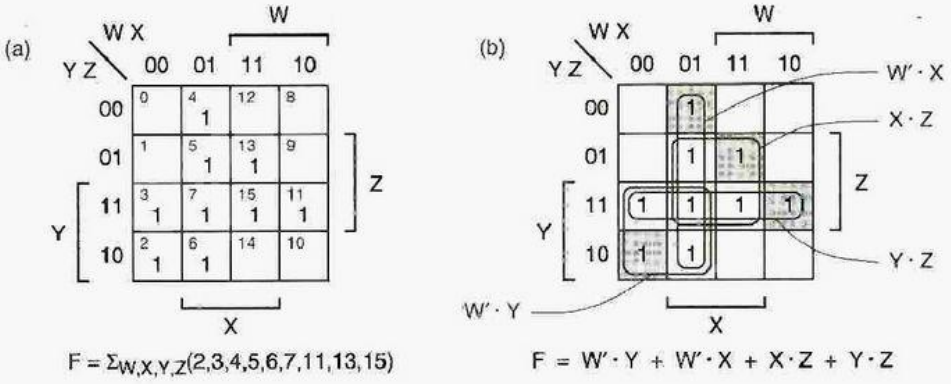


Figura 4-33 $F = \Sigma_{W,X,Y,Z}(2,3,4,5,6,7,11,13,15)$: (a) mapa de Karnaugh; b) implicantes primos y celdas 1 distinguidas.

tres de ellos. Así, ¿cómo podemos determinar sistemáticamente cuáles implicantes primos incluir y cuáles dejar fuera? Dos definiciones más son necesarias:

celda 1 distinguida

- Una *celda 1 distinguida* de una función lógica es una combinación de entrada que es cubierta por solamente un implicante primo.

implicante primo esencial

- Un *implicante primo esencial* de una función lógica es un implicante primo que cubre una o más celdas 1 distinguidas.

Puesto que un implicante primo esencial es el *único* implicante primo que cubre alguna celda 1, *debe* ser incluido en toda suma mínima para la función lógica. De este modo, el primer paso en el proceso de selección del implicante primo es simple: identificamos celdas 1 distinguidas y los correspondientes implicantes primos e incluimos los implicantes primos esenciales en la suma mínima. Entonces solamente necesitamos determinar cómo cubrir las celdas 1, si hay, que no estén cubiertas por los implicantes primos esenciales. En el ejemplo de la figura 4-32, las tres celdas 1 distinguidas se encuentran sombreadas, y los correspondientes implicantes primos esenciales se encuentran encerrados en un círculo con líneas más gruesas. Todas las celdas 1 en este ejemplo están cubiertas por implicantes primos esenciales, de manera que no necesitamos ir más allá. Del mismo modo, la figura 4-33 muestra un ejemplo donde todos los implicantes primos son esenciales, y así todos se incluyen en la suma mínima.

Una función lógica en la que no todas las celdas 1 se encuentran cubiertas por implicantes primos esenciales se ilustra en la figura 4-34. Al remover los implicantes primos esenciales y las celdas 1 que cubren, obtenemos un mapa reducido con solamente una sola celda 1 y 2 implicantes primos que la cubren. La elección en este caso es simple: hacemos uso del término de producto $W' \cdot Z$ porque tiene menos entradas y por consiguiente menor costo.

Para casos más complejos, necesitamos una definición más:

eclipse

- Dados dos implicantes primos P y Q en un mapa reducido, se dice que P eclipsa a Q (lo cual escribimos como P ... Q) si P cubre por lo menos todas las celdas 1 cubiertas por Q.

Si P no cuesta más que Q y eclipsa a Q, entonces al eliminar Q de la consideración no nos impide hallar una suma mínima; es decir, P es al menos tan bueno como Q.

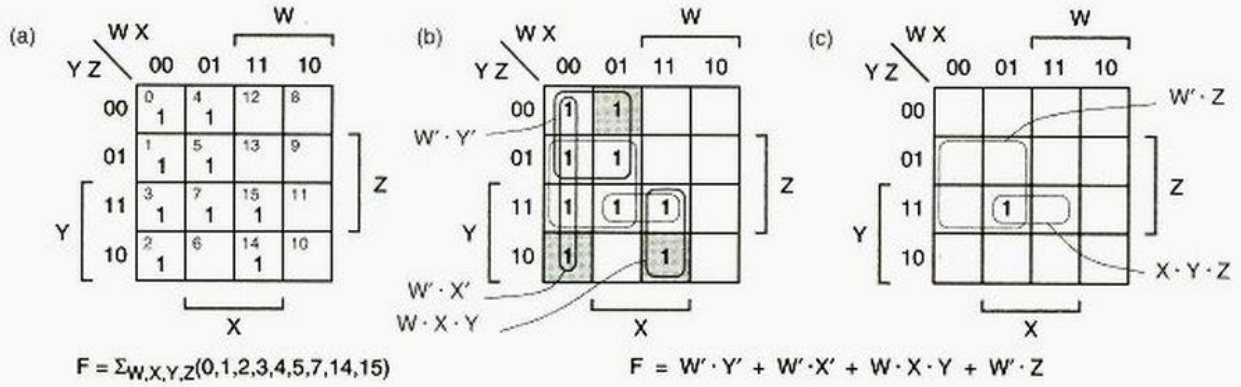


Figura 4-34 $F = \Sigma_{W,X,Y,Z}(0,1,2,3,4,5,7,14,15)$: (a) mapa de Karnaugh; b) implicantes primos y celdas 1 distinguidas; c) mapa reducido después de la eliminación de implicantes primos esenciales y celdas 1 cubiertas.

Un ejemplo de eclipsamiento se ilustra en la figura 4-35. Después de eliminar los implicantes primos esenciales nos quedamos con dos celdas 1, cada una de las cuales se encuentra cubierta por dos implicantes primos. Sin embargo, $X \cdot Y \cdot Z$ eclipsa los otros dos implicantes primos, los cuales por consiguiente se eliminan o remueven de nuestra consideración. Las dos celdas 1 son cubiertas entonces solamente por $X \cdot Y \cdot Z$, el cual es un *implicante primo esencial secundario* que debe incluirse en la suma mínima.

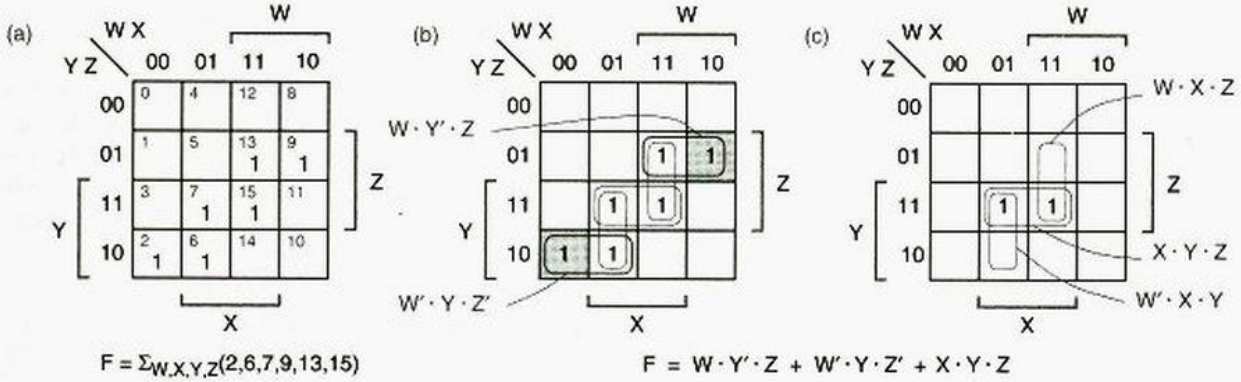
implicante primo esencial secundario

La figura 4-36 muestra un caso más difícil: una función lógica sin implicantes primos esenciales. Por ensayo y error encontramos dos diferentes sumas mínimas para esta función.

También podemos enfocar el problema sistemáticamente utilizando el método de ramificación. Comenzando con cualquier celda 1, arbitrariamente seleccionamos uno de los implicantes primos que la cubre, y lo incluimos como si fuera esencial. Esto simplifica

método de ramificación

Figura 4-35 $F = \Sigma_{W,X,Y,Z}(2,6,7,9,13,15)$: a) mapa de Karnaugh; b) implicantes primos y celdas 1 distinguidas; c) mapa reducido después de la eliminación de implicantes primos esenciales y celdas 1 cubiertas.



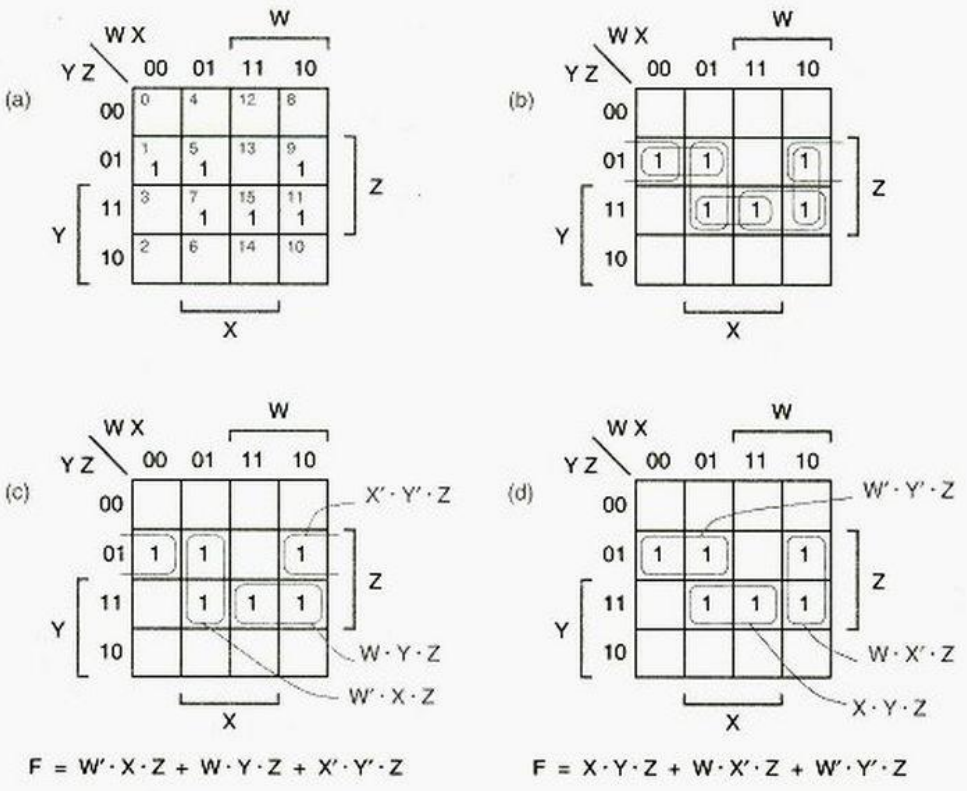


Figura 4-36 $F = \Sigma_{W,X,Y,Z}(1,5,7,9,11,15)$: a) mapa de Karnaugh; b) implicantes primos; c) una suma mínima; d) otra suma mínima.

el problema restante, el cual completamos de la forma habitual para encontrar una suma mínima tentativa. Repetimos este proceso comenzando con todos los otros implicantes primos que cubren la celda 1 de inicio, generando una diferente tentativa de suma mínima desde cada punto de partida. Es posible que nos atoremos con esto en el camino y tengamos que aplicar el método de ramificación de manera recursiva. Finalmente, examinamos todas las sumas mínimas tentativas que generamos de esta forma y seleccionamos una que verdaderamente sea mínima.

4.3.6 Simplificación de productos de sumas

Aplicando el principio de dualidad, minimizamos las expresiones de productos de sumas al examinar los ceros en un mapa de Karnaugh. Cada 0 en el mapa corresponde a un maxitérmino en el producto canónico de la función lógica. El proceso entero de la subsección anterior se reformula de una manera dual, incluyendo las reglas para escribir términos de suma correspondientes a los conjuntos en círculos de ceros, a fin de encontrar un *producto mínimo*.

producto mínimo

Afortunadamente, una vez que sabemos cómo hallar sumas mínimas, existe una manera más fácil de encontrar el producto mínimo para una función lógica dada F. El primer paso es complementar F para obtener F′. Suponiendo que F está expresada como

una lista de minitérminos o una tabla de verdad, la complementación es muy fácil: los unos de F′ son precisamente los ceros de F. A continuación, encontramos una suma mínima para F′ utilizando el método de la subsección anterior. Finalmente, complementamos el resultado empleando el teorema generalizado de DeMorgan, que produce un producto mínimo para (F′)′ = F. (Advierta que si simplemente "expandimos" la expresión de suma mínima para la función original, la expresión resultante del producto de sumas no es necesariamente mínima; por ejemplo, vea el ejercicio 4.61.)

En general, para hallar la realización de dos niveles de más bajo costo para una función lógica, tenemos que encontrar tanto una suma mínima como un producto mínimo y compararlos. Si una suma mínima para una función lógica tiene muchos términos, entonces un producto mínimo para la misma función puede tener menos términos. Como ejemplo trivial de este trueque, considere una función OR de 4 entradas:

$$F = (W) + (X) + (Y) + (Z) \text{ (una suma de cuatro términos de producto triviales)}$$
$$= (W + X + Y + Z) \text{ (un producto de un término de suma)}$$

Para un ejemplo no trivial, se le invita a encontrar el producto mínimo para la función que minimizamos en la figura 4-33 de la página 228; tiene apenas dos términos de suma.

La situación opuesta en ocasiones también es cierta, como se ilustra de manera trivial mediante una AND de 4 entradas:

$$F = (W) \cdot (X) \cdot (Y) \cdot (Z) \text{ (un producto de cuatro términos de suma triviales)}$$
$$= (W \cdot X \cdot Y \cdot Z) \text{ (una suma de un término de producto)}$$

Un ejemplo no trivial con un producto de sumas de mayor costo es la función en la figura 4-29 de la página 225.

Para algunas funciones lógicas, ambas formas mínimas son igualmente costosas. Por ejemplo, considere una función "OR exclusiva" de 3 entradas; ambas expresiones mínimas tienen cuatro términos y cada término tiene tres literales:

$$F = \Sigma_{X,Y,Z}(1,2,4,7)$$
$$= (X' \cdot Y' \cdot Z) + (X' \cdot Y \cdot Z') + (X \cdot Y' \cdot Z') + (X \cdot Y \cdot Z)$$
$$= (X + Y + Z) \cdot (X + Y' + Z') \cdot (X' + Y + Z') \cdot (X' + Y' + Z)$$

Todavía, en la mayoría de los casos, una forma o la otra dará mejores resultados. Examinar ambas formas es especialmente útil en los diseños basados en PLD.

MINIMIZACIÓN PLD Los PLD típicos tienen un arreglo AND-OR que corresponde a la forma de la suma de productos, de modo que se puede pensar que solamente la suma de productos mínima es importante para el diseño basado en PLD. Sin embargo, la mayoría de los PLD también tienen un inversor/búfer programable a la salida del arreglo AND-OR, que puede invertir o no. De esta forma, el PLD puede utilizar el equivalente de la suma mínima al emplear el arreglo AND-OR para realizar el complemento de la función deseada y entonces programar el inversor/búfer para invertir. La mayoría de los programas de minimización lógica para los PLD automáticamente encuentran tanto la suma mínima como el producto mínimo y seleccionan el que requiere de menos términos.

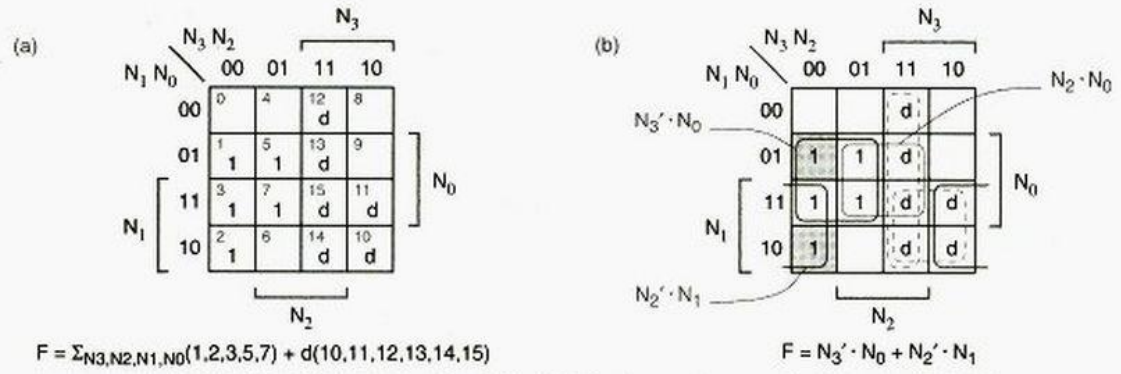


Figura 4-37 Detector de dígitos BCD primos: a) mapa de Karnaugh inicial; b) mapa de Karnaugh con implicantes primos y celdas 1 distinguidas.

*4.3.7 Combinaciones de entrada "sin importancia"

"sin importancia"

En ocasiones la especificación de un circuito combinacional es de tal modo que su salida no importa para ciertas combinaciones de entrada, denominadas *sin importancia*. Esto suele ser verdadero debido a que las salidas en realidad no importan cuando se presentan estas combinaciones de entrada, o porque estas combinaciones de entrada nunca se presentan durante el funcionamiento normal. Por ejemplo, supongamos que queremos construir un detector de números primos cuya entrada de 4 bits $N = N_3N_2N_1N_0$ sea siempre un dígito BCD; entonces los minitérminos 10-15 no deben presentarse nunca. Una función de detector de dígitos primos BCD puede por consiguiente escribirse como sigue:

$$F = \Sigma_{N_3,N_2,N_1,N_0}(1,2,3,5,7) + d(10,11,12,13,14,15)$$

conjunto d

La lista d(...) especifica las combinaciones de entrada sin importancia[1] para la función, también conocidas como el *conjunto d*. Aquí, F debe ser 1 para combinaciones de entrada en el conjunto de iniciación (1, 2, 3, 5, 7), F puede tener cualquier valor para las entradas en el conjunto d (10, 11, 12, 13, 14, 15), y F debe ser 0 para todas las otras combinaciones de entrada (en el conjunto 0).

La figura 4-37 muestra cómo hallar una realización de suma de productos mínima para el detector de dígitos BCD primos, incluyendo aquellos sin importancia. Las d en el mapa denotan las combinaciones de entrada sin importancia. Modificamos el procedimiento para encerrar en círculos conjuntos de unos (implicantes primos) como sigue:

- Permita que las d estén incluidas cuando se encierren en círculos conjuntos de unos, para hacer conjuntos tan grandes como sea posible. Esto reduce el número de variables en los implicantes primos correspondientes. Dos de esos implicantes primos ($N_2 \cdot N_0$ y $N_2' \cdot N_1$) aparecen en el ejemplo.
- No encierre en círculos ningún conjunto que contenga solamente las d. Incluir el término de producto correspondiente en la función incrementaría de forma innecesaria su costo. Dos de tales términos de producto ($N_3 \cdot N_2$ y $N_3 \cdot N_1$) se encuentran encerrados en círculos en el ejemplo.
- Sólo un recordatorio: como es costumbre, no encierre en círculos ninguno de los ceros.

*En este libro, las secciones opcionales están marcadas con un asterisco.

[1]Del inglés don't care, N. del T.

El resto del procedimiento es el mismo. En particular, buscamos celdas 1 distinguidas y *no* celdas d distinguidas, e incluimos solamente los correspondientes implicantes primos esenciales y cualquier otro que sea necesario para cubrir todos los unos en el mapa. En la figura 4-37, los dos implicantes primos esenciales son suficientes para cubrir todos los unos en el mapa. Dos de los d también son cubiertos, de modo que F será 1 para las combinaciones de entrada sin importancia 10 y 11, y 0 para las otras sin importancia.

Algunos HDL, incluyendo ABEL, proporcionan los medios para que el diseñador especifique las entradas sin importancia, y el programa de minimización lógica toma éstos en cuenta cuando calcula una suma mínima.

*4.3.8 Minimización de salida múltiple

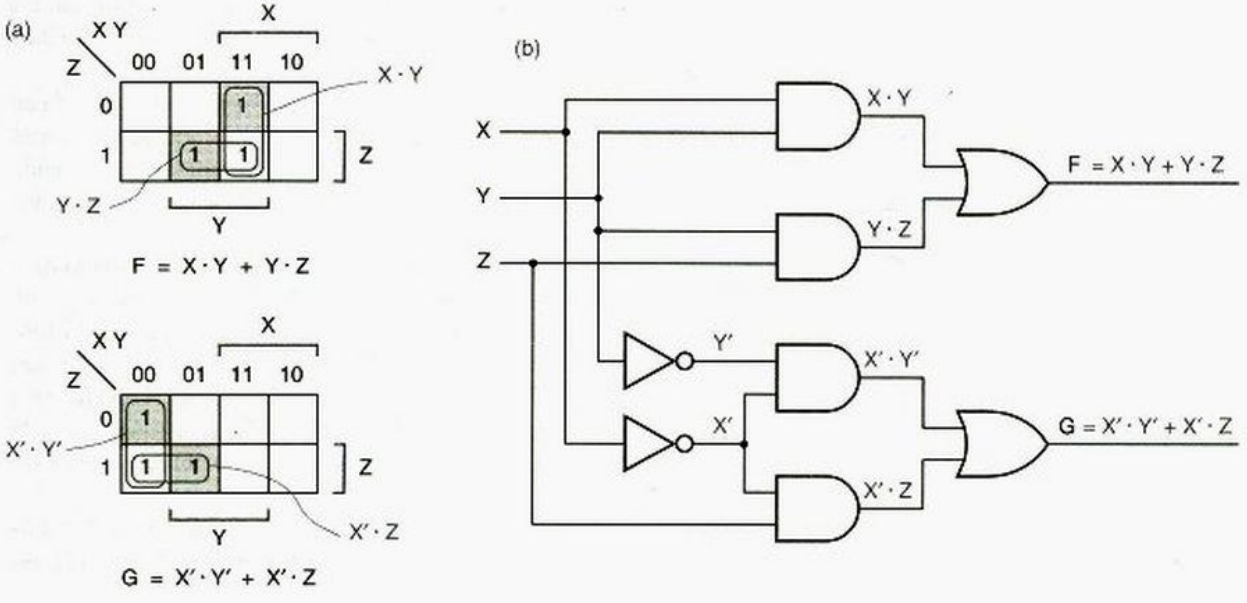
La mayoría de los circuitos lógicos combinacionales prácticos requieren de más de una salida. Siempre podemos manejar un circuito con n salidas como n problemas de diseño independientes de salida simple. Sin embargo, al hacerlo así, perdemos algunas oportunidades de optimización. Por ejemplo, considere las dos siguientes funciones lógicas:

$$F = \Sigma_{X,Y,Z}(3,6,7)$$

$$G = \Sigma_{X,Y,Z}(0,1,3)$$

La figura 4-38 ilustra el diseño de F y G como dos funciones independientes de salida simple. Sin embargo, como se muestra en la figura 4-39, también podemos hallar un par de expresiones de suma de productos que comparten un término de producto, de tal modo que el circuito resultante tiene una compuerta menos que nuestro diseño original.

Figura 4-38 Tratamiento de un diseño de 2 salidas como dos diseños independientes de salida simple: a) mapas de Karnaugh; b) circuito "mínimo".



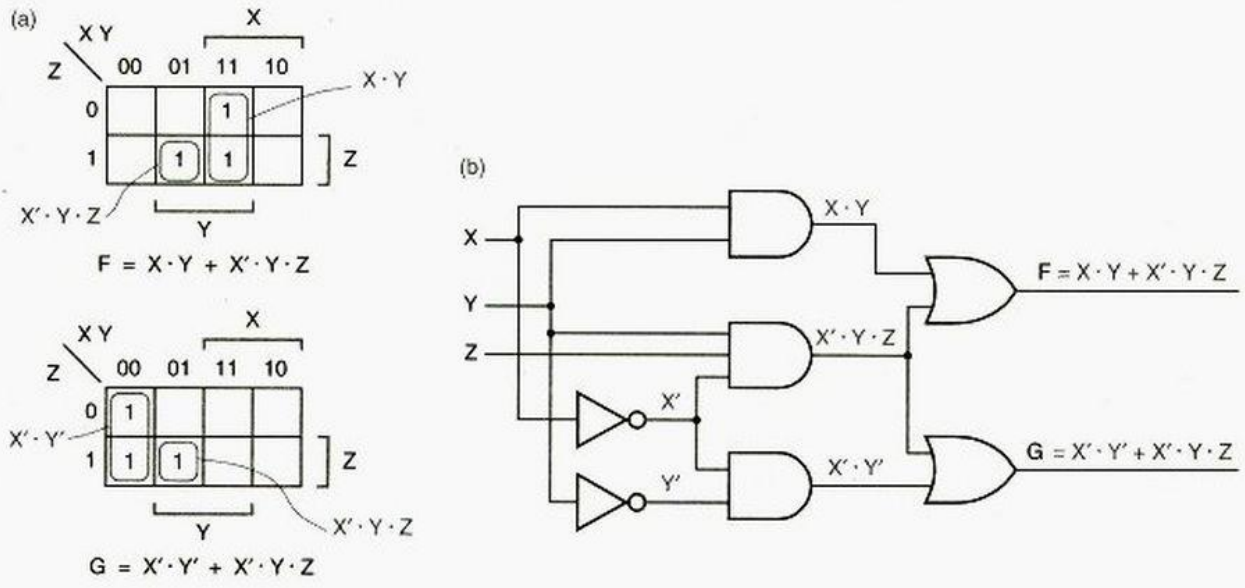


Figura 4-39 Minimización de salida múltiple para un circuito de 2 salidas: a) mapas minimizados incluyendo un término compartido; b) circuito de salida múltiple mínimo.

Cuando diseñamos circuitos combinacionales de salida múltiple utilizando compuertas discretas, como en un ASIC, el compartir términos de producto obviamente reduce el tamaño y costo del circuito. Además, los PLD contienen múltiples copias de la estructura de suma de productos que hemos estado aprendiendo cómo minimizar, uno por salida, algunos PLD permiten que términos de producto se compartan entre salidas múltiples. De este modo, las ideas presentadas en esta subsección se emplean en muchos programas de minimización lógica.

Probablemente usted haya "echado un ojo" a los mapas de Karnaugh para F y G en la figura 4-39 y haya descubierto la solución mínima. Sin embargo, los circuitos más grandes se minimizan solamente con un algoritmo formal de minimización de salida múltiple. Daremos una idea general de las ideas de un algoritmo de esta clase aquí; los detalles se encuentran en la sección de Referencias.

función de producto m

La clave para el éxito de la minimización de salida múltiple de un conjunto de n funciones es considerar no solamente las n funciones de salida simple originales, sino también las "funciones producto". Una *función de producto m* de un conjunto de n funciones es el producto de m de las funciones, donde $2 \leq m \leq n$. Existen $2^n - n - 1$ de esas funciones. Afortunadamente, $n = 2$ en nuestro ejemplo y tenemos solamente una función producto, F· G, a considerar. Los mapas de Karnaugh para F, G y F · G se muestran en la figura 4-40; en general, el mapa para una función de producto m se obtiene al unir los mapas de sus m componentes con AND.

implicante primo de salida múltiple

Un *implicante primo de salida múltiple* de un conjunto de n funciones es un implicante primo de una de las n funciones o de una de las funciones producto. El primer paso

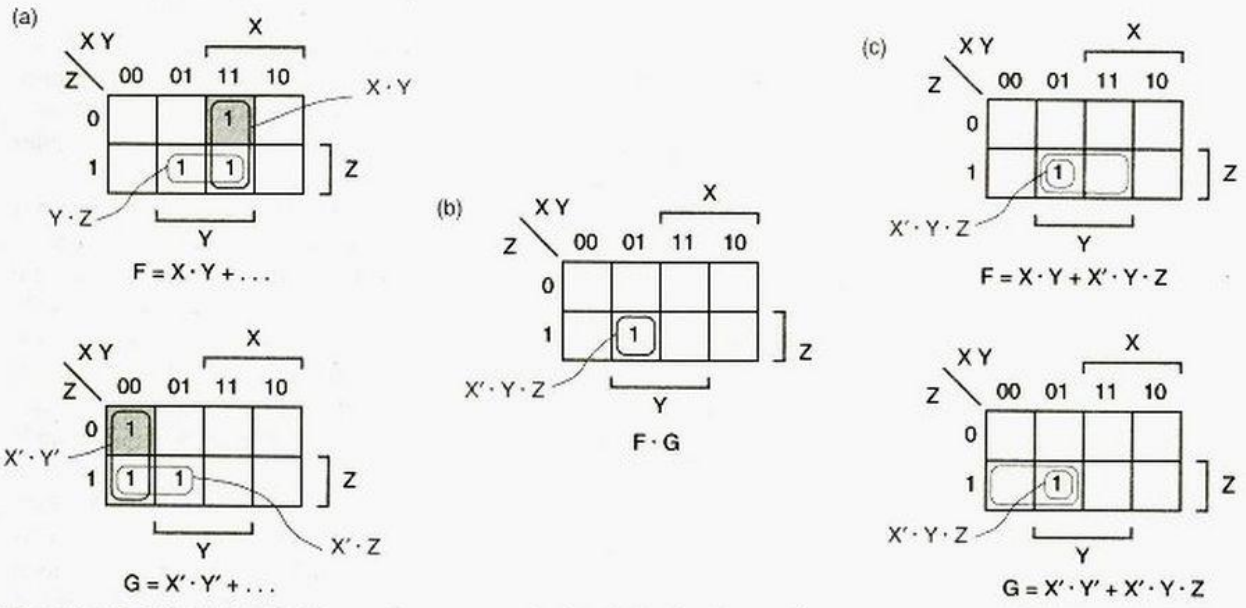


Figura 4-40 Mapas de Karnaugh para un conjunto de dos funciones: a) mapas para F y G; b) mapa de producto 2 para F · G; c) mapas reducidos para F y G después de la eliminación de los implicantes primos esenciales y celdas 1 cubiertas.

en la minimización de salida múltiple es encontrar todos los implicantes primos de salida múltiple. Cada implicante primo de una función de producto *m* es un posible término a incluir en las correspondientes salidas *m* del circuito. Si intentáramos minimizar un conjunto de 8 funciones, tendríamos que hallar los implicantes primos para $2^8 - 8 - 1 = 247$ funciones de producto además de para las 8 funciones dadas. ¡Obviamente, la minimización de salida múltiple no es para los timoratos!

Una vez que hemos hallado los implicantes primos de salida múltiple, intentemos simplificar el problema identificando los que son esenciales. Una *celda 1 distinguida* de una función F particular de salida simple es una celda 1 que está cubierta exactamente por implicante primo de F o de las funciones de producto que involucran F. Las celdas 1 distinguidas en la figura 4-40(a) se encuentran sombreadas. Un *implicante primo esencial* de una función de salida simple particular es aquella que contiene una celda 1 distinguida. Como en la minimización de salida simple, los implicantes primos esenciales deben incluirse en una solución de mínimo costo. Solamente las celdas 1 que no están cubiertas por implicantes primos esenciales se consideran en el resto del algoritmo.

celda 1 distinguida

implicante primo esencial

El paso final es seleccionar un conjunto mínimo de implicantes primos para cubrir las celdas 1 restantes. En este paso debemos considerar todas las *n* funciones de manera simultánea, incluyendo la posibilidad de compartir; los detalles de este procedimiento se discuten en las Referencias. En el ejemplo de la figura 4-40(c), vemos que existe un término de producto simple compartido que cubre la celda 1 restante tanto en F como en G.

*4.4 Métodos de minimización programados

Obviamente, la minimización lógica es un proceso muy complicado. En las aplicaciones de diseño lógico real usted probablemente encontrará dos clases de problemas de minimización: funciones de unas cuantas variables que se examinan utilizando los métodos de la sección anterior, y funciones de salida múltiple más complejas que son imposibles sin el uso de un programa de minimización.

Sabemos que la minimización se efectúa visualmente para funciones con unas cuantas variables utilizando el método del mapa de Karnaugh. En esta sección demostraremos que es posible realizar las mismas operaciones para funciones con un número arbitrariamente grande de variables (por lo menos, en principio) empleando un método de tabulación llamado el *algoritmo de Quine-McCluskey*. Como todos los algoritmos, el algoritmo de Quine-McCluskey se traduce a un programa de computadora. Y como el método del mapa, el algoritmo tiene dos pasos: (a) encontrar todos los implicantes primos de la función, y (b) seleccionar un conjunto mínimo de implicantes primos que cubra la función.

algoritmo de Quine-McCluskey

El algoritmo de Quine-McCluskey se describe a menudo en términos de tablas escritas a mano y procedimientos de verificación manual. Sin embargo, ya que nadie emplea estos procedimientos manualmente, es más apropiado para nosotros discutir el algoritmo en términos de estructuras de datos y funciones en un lenguaje de programación de alto nivel. El objetivo de esta sección es proporcionarle una evaluación de la complejidad computacional involucrada en un problema de minimización grande. Consideramos sólo funciones de salida simple, completamente especificadas; las funciones de salidas múltiple y sin importancia se manejan mediante modificaciones bastante directas a los algoritmos de salida simple, como se discute en las Referencias.

*4.4.1 Representación de términos de producto

El punto de partida para el algoritmo de minimización de Quine-McCluskey es la tabla de verdad o, de manera equivalente, la lista de minitérminos de una función. Si la función se especifica de manera diferente, primero debe ser convertida a esta forma. Por ejemplo, una expresión lógica de n variables arbitraria se expande multiplicándola (quizás utilizando el teorema de DeMorgan en el proceso) para obtener una expresión de suma de productos. Una vez que tenemos una expresión de suma de productos, cada término de producto de p variables produce 2^{n-p} minitérminos en la lista de minitérminos.

Mostramos en la sección 4.1.6 que un minitérmino de una función lógica de n variables se representa por un entero de n bits (el número minitérmino), donde cada bit indica si la variable correspondiente se encuentra complementada o no complementada. Sin embargo, un algoritmo de minimización también debe tratar con términos de producto que no son minitérminos, donde algunas variables no aparecen en absoluto. De este modo, debemos representar tres posibilidades para cada variable en un término de producto general:

1 Sin complementar.

0 Complementada.

x No aparece.

Estas posibilidades están representadas por una cadena de n de los dígitos anteriores en la *representación de cubo* de un término de producto. Por ejemplo, si estamos trabajando con términos de producto de hasta ocho variables, X7, X6, ..., X1, X0, escribimos los siguientes términos de producto y sus representaciones de cubo:

representación de cubo

$$X7' \cdot X6 \cdot X5 \cdot X4' \cdot X3 \cdot X2 \cdot X1 \cdot X0' \equiv 01101110$$
$$X3 \cdot X2 \cdot X1 \cdot X0' \equiv xxxx1110$$
$$X7 \cdot X5' \cdot X4 \cdot X3 \cdot X2' \cdot X1 \equiv 1x01101x$$
$$X6 \equiv x1xxxxxx$$

Note que, por conveniencia, nombramos las variables justo como las posiciones de los bits en enteros binarios de n bits.

Desde el punto de vista de la nomenclatura de cubo n y subcubo m de la sección 2.14, la cadena 1x01101x representa un subcubo 2 de un cubo 8, y la cadena 01101110 representa un subcubo 0 de un cubo 8. Sin embargo, en la literatura de minimización, la máxima dimensión n de un cubo o subcubo está por lo regular implícita, y un subcubo m se denomina simplemente un "cubo m" o un "cubo" para abreviar; seguiremos esta práctica en esta sección.

Para representar un término de producto en un programa de computadora, utilizamos una estructura de datos con n elementos, cada elemento tiene tres valores posibles. En C, podemos hacer las declaraciones siguientes:

```
typedef enum {complemented, uncomlemented, doesntappear} TRIT;
typedef TRIT[16] CUBE;   /*  Represents a single product
                             therm with up to 16 variables */
```

Sin embargo, estas declaraciones no conducen a una representación interna particularmente eficiente de los cubos. Como veremos, los cubos son más fáciles de manipular empleando instrucciones convencionales de computadora si un término de producto de n variables es representado por dos palabras de computadora de n bits, como se sugiere por las declaraciones siguientes:

```
#define MAX_VARS 16   /* Máx # of variables in a product term */
typedef unsigned shor WORD;   /* Use 16-bit words /*
struct cube {
  WORD t; /* Bits 1 for uncomplemented variables. */
  WORD f; /* Bits 1 for complemented variable. */
};
typedef struct cube CUBE;
CUBE P1, P2, P3;   /* Allocate three cubes for use by program
```

Aquí, WORD es un entero de 16 bits, y un término de producto de 16 variables se representa mediante un registro (record) con dos WORDs, como se muestra en la figura 4-41(a). La primera palabra en un CUBE tiene 1 para cada variable en el término de producto que aparece sin complementar (o "verdadero", t = "true") y la segunda tiene un 1 para cada variable que aparece complementada (o "falsa", f), Si una posición particular de bit tiene ceros en ambas WORD, entonces la variable correspondiente no

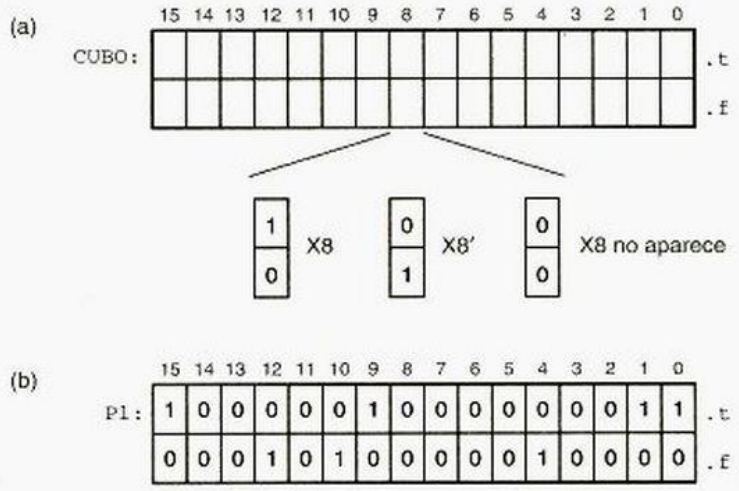


Figura 4-41
Representación interna de términos de producto de 16 variables en un programa Pascal: a) formato general; (b) P1 = X15·X12′·X10′·X9·X4′·X1·X0

aparece, mientras que el caso de una posición de bit particular teniendo unos en ambas WORD no se usa. De este modo, la variable del programa P1 en b) representa el término de producto **P1 = X15 · X12′ · X10′ · X9 · X4′ · X1 · X0**. Si deseamos representar una función lógica F de hasta 16 variables, que contenga hasta 100 términos de producto, declaramos un arreglo de 100 CUBEs:

```
CUBE F[100];  /* Storage for a logic function
                 with up to 100 product terms. */
```

Aplicando la representación de cubo precedente, es posible escribir funciones de C breves y eficientes que manipulen términos de producto en formas útiles. La tabla 4-8 muestra varias de estas funciones. Correspondiente a dos de las funciones, la figura 4-42 retrata cómo dos cubos se comparan y combinan si es posible empleando el teorema T10.

Figura 4-42 Manipulaciones de cubo: a) determinación de si dos cubos son combinables utilizando el teorema T10, término · X + término · X′ = término; b) combinación de cubos empleando el teorema T10.

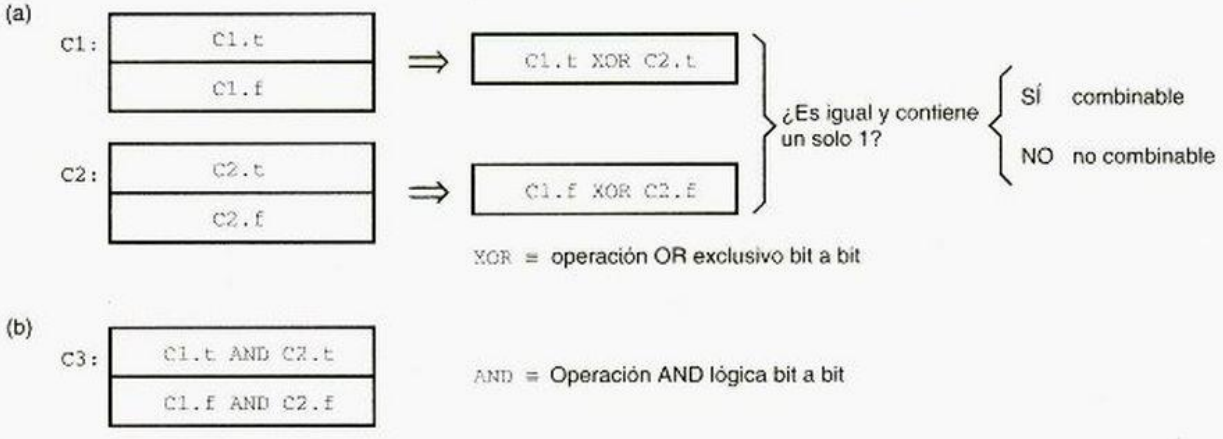


Tabla 4-8 Comparación de cubo y funciones de combinación utilizadas en programa de minimización.

```
int EqualCubes(CUBE C1, CUBE C2)        /* Returns true if C1 and C2 are identical. */
{
  return ( (C1.t == C2.t) && (C1.f == C2.f) );
}

int Oneone(WORD w)          /* Returns true if w has exactly one 1 bit .              */
{                           /* Optimizing the speed of this routine is critical       */
  int ones, b;              /*   and is left as an exercise for the hacker            */
  ones = 0;
  for (b=0; b<MAX_VARS; b++) {
    if (w & 1) ones++;
    w = w>>1;
  }
  return((ones==1));
}

int Combinable(CUBE C1, CUBE C2)
{                           /* Returns true if C1 and C2 differ in only one variable, */
  WORD twordt, twordf;      /* which appears true in one and false in the other.      */

  twordt = C1.t ^ C2.t;
  twordf = C1.f ^ C2.f;
  return( (twordt==twordf) && Oneone(twordt) );
}

void Combine(CUBE C1, CUBE C2, CUBE *C3)
                          /* Combines C1 and C2 using theorem T10, and stores the    */
{                           /*    result in C3. Assumes Combinable(C1,C2) is true.      */
  C3->t = C1.t & C2.t;
  C3->f = C1.f & C2.f;
}
```

término · X + término · X′ = término. este teorema nos dice que dos términos de producto se combinan si difieren en solamente una variable que aparece complementada en un término y no complementada en el otro. La combinación de dos cubos m produce un cubo $(m + 1)$. Utilizando la representación de cubo, apliquemos el teorema de combinación a unos cuantos ejemplos:

$$010 + 000 = 0x0$$

$$00111001 + 00111000 = 0011100x$$

$$101xx0x0 + 101xx1x0 = 101xxxx0$$

$$x111xx00110x000x + x111xx00010x000x = x111xx00x10x000x$$

*4.4.2 Cálculo de implicantes primos mediante la combinación de términos de producto

El primer paso en el algoritmo de Quine-McCluskey es determinar todos los implicantes primos de la función lógica. Con un mapa de Karnaugh, hacemos esto visualmente al identificar "los conjuntos rectangulares más grandes posibles de 1s". En el algoritmo, esto se hace mediante la sistemática y repetida aplicación del teorema T10 para combinar minitérminos, luego cubos 1, cubos 2 y así sucesivamente, creando los cubos más grandes posibles (términos de producto más pequeños posibles) que cubran solamente unos de la función.

El programa C en la tabla 4-9 se aplica al algoritmo para funciones con hasta 16 variables. Utiliza arreglos bidimensionales, `cubes[m][j]` y `covered[m][j]`, para seguir la pista de `MAX_VARS` cubos *m*. Los cubos 0 (minitérminos) son suministrados por el usuario. Comenzando con los cubos 0, el programa examina cada par de cubos en cada nivel y los combina cuando es posible en cubos al siguiente nivel. Los cubos que se combinan en un cubo del siguiente nivel se marcan como "covered" ("cubiertos"); los cubos que no están cubiertos son implicantes primos.

Aun cuando el programa de la tabla 4-9 es corto, un programador experimentado podría llegar a pensar de manera pesimista sólo viendo su estructura. El ciclo `for` interno se encuentra anidado a cuatro niveles de profundidad, y el número de veces que se ejecuta es del orden de `MAX_VARS` · `MAX_CUBES`^{3}. ¡Está bien, eso es un exponente, no un indicador de nota al pie! Elegimos el valor `maxCubes` = 100 algo arbitrariamente (de hecho, con demasiado optimismo para muchas funciones), pero si usted cree en este número, entonces el ciclo interno se ejecuta *millares y millares de millones* de veces.

El máximo número de minitérminos de una función de n variables es 2^n, por supuesto, y así por todo derecho el programa en la tabla 4-9 debe declarar `maxCubes` como 2^{16}, al menos para manejar el máximo número posible de cubos 0. Una declaración así no será demasiado pesimista. Si una función de n variables tiene un término de producto igual a una variable simple, entonces 2^{n-1} minitérminos son de hecho necesarios para cubrir ese término de producto.

Para cubos mayores, la situación es en realidad peor. El número de posibles subcubos m de un cubo n es $\binom{n}{m} \times 2^{n-m}$, donde el coeficiente binomial $\binom{n}{m}$ es el número de maneras de elegir m variables que sean x's, y 2^{n-m} es el número de maneras de asignar ceros y unos a las variables restantes. Para funciones de 16 variables, el peor caso se presenta con $m = 5$; existen 8,945,664 posibles subcubos 5 de un cubo 16. El número total de subcubos m distintos de un cubo n, sobre todos los valores de m es 3^n. De esta forma, un programa de minimización general requiere *mucho* más memoria que la que hemos asignado en la tabla 4-9.

Existen varias acciones que podemos tomar para optimizar el espacio de almacenamiento y el tiempo de ejecución requerido en la tabla 4-9 (vea los ejercicios 4.77-4.80), pero son insignificantes en comparación a la aplastante complejidad combinatoria del problema. Así, incluso con las rápidas computadoras de la actualidad y sus inmensos recursos de memoria, la aplicación directa del algoritmo de Quine-McCluskey para generar implicantes primos se encuentra generalmente limitada a funciones con solamente unas cuantas variables (menores de 15-20).

Tabla 4-9 Un programa en lenguaje C que encuentra los implicantes primos con la ayuda del algoritmo de Quine-McCluskey.

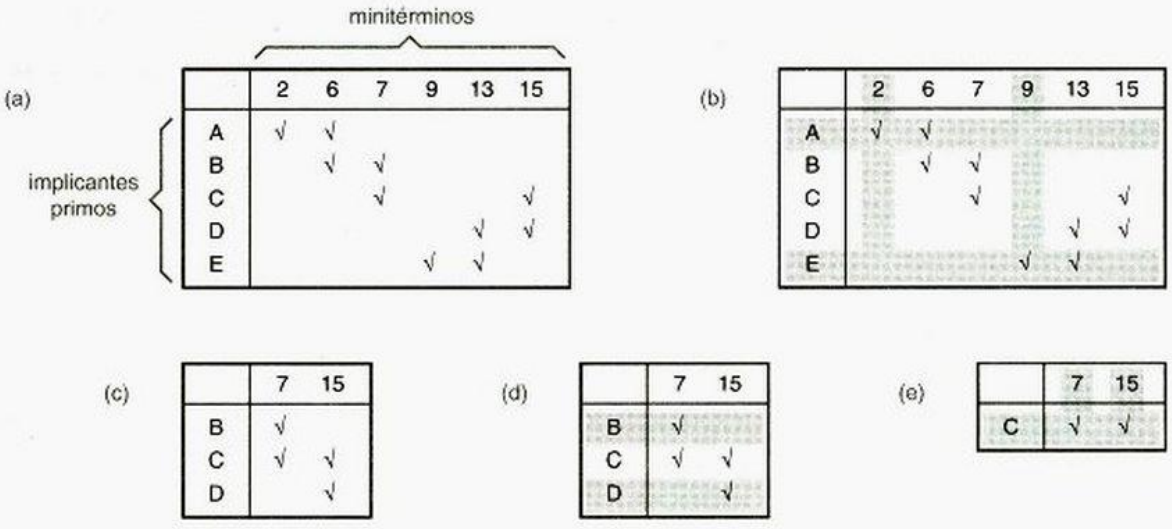
```
#define TRUE   1
#define FALSE  0
#define MAX_CUBES 50

void main()
{
  CUBE cubes[MAX_VARS+1][MAX_CUBES];
  int covered[MAX_VARS+1][MAX_CUBES];
  int numCubes[MAX_VARS+1];
  int m;          /* Value of m in an m-cube, i.e., ''level m''. */
  int j, k, p;    /* Indices into the cubes or covered array.    */
  CUBE tempCube;
  int found;

  /* Initialize number of m-cubes at each level m. */
  for (m=0; m<MAX_VARS+1; m++) numCubes[m] = 0;

  /* Read a list of minterms (0-cubes) supplied by the user, storing them    */
  /* in the cubes[0,j] subarray, setting covered[0,j] to false for each      */
  /* minterm, and setting numCubes[0] to the total number of minterms read.  */
ReadMinterms;

  for (m=0; m<MAX_VARS; m++)            /* Do for all levels except the last */
    for (j=0; j<numCubes[m]; j++)       /* Do for all cubes at this level    */
      for (k=j+1; k<numCubes[m]; k++)   /* Do for other cubes at this level  */
        if (Combinable(cubes[m][j], cubes[m][k])) {
          /* Mark the cubes as covered. */
          covered[m][j] = TRUE;  covered[m][k] = TRUE;
          /* Combine into an (m+1)-Cube, store in tempCube. */
          Combine(cubes[m][j], cubes[m][k], &tempCube);
          found = FALSE;  /* See if we've generated this one before. */
          for (p=0; p<numCubes[m+1]; p++)
            if (EqualCubes(cubes[m+1][p],tempCube)) found = TRUE;
          if (!found) {  /* Add the new cube to the next level. */
            numCubes[m+1] = numCubes[m+1] + 1;
            cubes[m+1][numCubes[m+1]-1] = tempCube;
            covered[m+1][numCubes[m+1]-1] = FALSE;
          }
        }
  for (m=0; m<MAX_VARS; m++)        /* Do for all levels              */
    for (j=0; j<numCubes[m]; j++)   /* Do for all cubes at this level */
      /* Print uncovered cubes -- éstos son los implicantes primos. */
      if (!covered[m][j]) PrintCube(cubes[m][j]);
}
```



	2	6	7	9	13	15
A	√	√				
B		√	√			
C			√			√
D					√	√
E				√	√	

	2	6	7	9	13	15
A	√	√				
B		√	√			
C			√			√
D					√	√
E				√	√	

	7	15
B	√	
C	√	√
D		√

	7	15
B	√	
C	√	√
D		√

	7	15
C	√	√

Figura 4-43 Tablas de implicantes primos: a) tabla original; b) con celdas 1 distinguidas e implicantes primos esenciales; c) después de la eliminación de los implicantes primos esenciales; d) con los renglones eclipsados; e) después de la eliminación de los renglones eclipsados, con los implicantes primos esenciales secundarios.

*4.4.3 Encontrando una cubierta mínima empleando una tabla de implicantes primos

tabla de implicantes primos

El segundo paso al minimizar una función lógica combinacional, una vez que tenemos una lista de todos sus implicantes primos, es seleccionar un subconjunto mínimo de ellos para cubrir todos los unos de la función. El algoritmo de Quine-McCluskey utiliza un arreglo bidimensional conocido como una *tabla de implicantes primos* para hacer esto. La figura 4-43(a) muestra una pequeña pero representativa tabla de implicantes primos, correspondiente al problema de minimización de mapa de Karnaugh de la figura 4-35. Hay una columna por cada minitérmino de la función, y un renglón por cada implicante primo. Cada entrada es un bit que es 1, si y sólo si, el implicante primo para ese renglón cubre el minitérmino para esa columna (mostrado en la tabla como una "palomita").

Los pasos para seleccionar los implicantes primos con la tabla son análogos a los pasos que se utilizan en la sección 4.3.5 con los mapas de Karnaugh:

1. Identifique las celdas 1 distinguidas. Éstas son fácilmente identificadas en la tabla como columnas con un solo 1, como se muestra en la figura 4-43(b).
2. Incluya todos los implicantes primos esenciales en la suma mínima. Un renglón que contiene una palomita en una o más columnas de celdas 1 distinguidas corresponde a un implicante primo esencial.
3. Elimine de su consideración los implicantes primos esenciales y las celdas 1 (minitérminos) que ellos cubran. En la tabla, esto se hace eliminando los renglones y las columnas correspondientes, marcados en color gris en la figura 4-43(b). Si cualquiera de los renglones no tienen palomitas restantes, entonces también se eliminan;

los correspondientes implicantes primos son *redundantes*, es decir, completamente cubiertos por implicantes primos esenciales. Este paso nos lleva a la tabla reducida que se muestra en (c).

implicante primo redundante

4. Elimine de su consideración cualquier implicante primo que esté "eclipsado" por otros con costo menor o igual. En la tabla, esto se hace al borrar cualquier renglón cuyas columnas con palomitas sean un subconjunto propio de otros renglones, y al eliminar todos menos uno de un conjunto de renglones con idénticas columnas con palomitas. Esto se muestra sombreado en (d) y conduce a la tabla aún más reducida en (e).
   Cuando una función se realiza en un PLD, todos sus implicantes primos se consideran como si tuvieran igual costo, debido a que todas las compuertas AND en un PLD tienen todas sus entradas disponibles. De otro modo, los implicantes primos deben ser clasificados y seleccionados de acuerdo al número de entradas de compuerta AND.
5. Identifique las celdas 1 distinguidas e incluya todos los implicantes primos esenciales secundarios en la suma mínima. Como antes, cualquier renglón que contenga una palomita en una o más columnas de celdas 1 distinguidas corresponde a un implicante primo esencial secundario.
6. Si todas las columnas restantes están cubiertas por los implicantes primos esenciales secundarios, como en e), habremos terminado. De otro modo, si cualquier implicante primo esencial secundario se encontrara en el paso anterior, regresaremos al paso 3 y repetiremos el ciclo (iteración). De otra manera, deberá utilizar el método de ramificación, como se describió en la sección 4.3.5. Esto involucra la elección de los renglones, uno a la vez, tratándolos como si fueran esenciales, y volviendo a pasar (y pasando) por los pasos 3 a 6.

Aunque una tabla de implicantes primos permite un algoritmo de selección bastante directo de implicantes primos, la estructura de datos que requiere un programa de computadora es inmensa, puesto que requiere del orden de $p \cdot 2^n$ bits, donde p es el número de implicantes primos y n es el número de bits de entrada (suponiendo que la función dada produzca una salida 1 para la mayoría de las combinaciones de entrada). Peor aún, ejecutar los pasos que hemos descrito alegremente con unas cuantas frases hace un momento requiere de una inmensa cantidad de cálculos.

*4.4.4 Otros métodos de minimización

Aunque las anteriores subsecciones forman una introducción a los algoritmos de minimización lógica, los métodos que describen no son de ningún modo los últimos ni los más grandes. Presionados por la siempre creciente densidad de los chips VLSI, muchos investigadores han descubierto maneras más efectivas para minimizar funciones lógicas combinacionales. Sus resultados caen aproximadamente en tres categorías:

1. *Mejoras computacionales*. Los algoritmos mejorados utilizan por lo regular estructuras de datos más inteligentes o reacomodan el orden de los pasos para reducir los requerimientos de memoria y el tiempo de ejecución de los algoritmos clásicos.
2. *Métodos heurísticos*. Algunos problemas de minimización son demasiado grandes para resolverse utilizando un algoritmo "exacto". Estos problemas se resuelven

utilizando atajos y conjeturas bien planteados, que ayudan a reducir el tamaño de la memoria y el tiempo de ejecución a una fracción del tiempo que requiere un algoritmo exacto. Sin embargo, más que hallar una expresión probablemente mínima para una función lógica, los métodos heurísticos intentan encontrar una "casi mínima".

Incluso para problemas que pueden ser resueltos mediante un método "exacto", un método heurístico por lo regular encuentra una buena solución diez veces más rápido: El programa heurístico con más éxito, Espresso-II, de hecho produce resultados mínimos o casi mínimos para la mayor parte de los problemas (dentro de uno o dos términos de producto), incluyendo problemas con docenas de entradas y cientos de términos de producto.

3. *Examen de las cosas desde un enfoque diferente.* Como lo mencionamos con anterioridad, la minimización de salida múltiple puede manejarse mediante modificaciones directas (que son bastante mecánicas) de los métodos de minimización de salida simple. Sin embargo, al examinar la minimización de salida múltiple como un problema en lógica multivaluada (no binaria), los diseñadores del algoritmo Espresso-MV fueron capaces de hacer mejoras sustanciales en el desempeño por arriba de Espresso-II.

Puede encontrar información adicional de estos métodos en la sección de referencias.

*4.5 Riesgos en el tiempo

comportamiento de estado estacionario

Los métodos de análisis que hemos desarrollado en la sección 4.2 ignoran el retardo de circuito y predicen solamente el *comportamiento de estado estacionario* de los circuitos lógicos combinacionales. Es decir, predicen una salida de circuito como una función de sus entradas bajo la suposición de que las entradas han estado estables por largo tiempo, relativo a los retardos en la electrónica del circuito. Sin embargo, mostramos en la sección 3.6 que el retardo real desde un cambio en la entrada hasta el correspondiente cambio en la salida en un circuito lógico real es distinto de cero y depende de muchos factores.

comportamiento transitorio

glitch o interferencia

riesgo

Debido a los retardos del circuito, el *comportamiento transitorio* de un circuito lógico puede diferir del que se predice mediante un análisis de estado estacionario. En particular, una salida de circuito puede producir un pulso corto, frecuentemente llamado un "glitch" interferencia a un tiempo cuando el análisis de estado estacionario predice que la salida no debería cambiar. Se dice que existe un *riesgo* cuando un circuito tiene la posibilidad de producir una interferencia de esa clase. Que en realidad ocurra o no una interferencia depende de los retardos exactos y otras características eléctricas del circuito. Puesto que tales parámetros son difíciles de controlar en circuitos de producción, un diseñador lógico debe estar preparado para eliminar los riesgos (la *posibilidad* de una interferencia) aun cuando pueda ocurrir una interferencia solamente bajo una combinación en el peor de los casos de condiciones lógicas y eléctricas.

*4.5.1 Riesgos estáticos

riesgo de 1 estático

Un *riesgo de 1 estático* es la posibilidad de que una salida del circuito produzca una interferencia 0 cuando esperaríamos que la salida permaneciera en un sutil 1 estacionario, con base en un análisis estático de la función del circuito. Una definición formal se da a continuación.

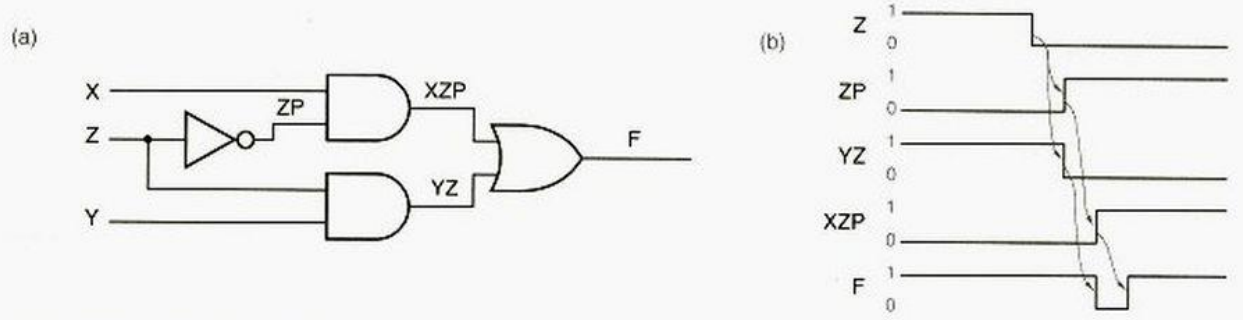


Figura 4-44 Circuito con un riesgo de 1 estático: a) diagrama lógico; b) diagrama de temporización.

Definición: Un riesgo de 1 estático es un par de combinaciones de entrada que: a) difieren en solamente una variable de entrada y b) ambas proporcionan una salida 1; tal que es posible que ocurra una salida 0 momentánea durante una transición en la variable de entrada que difiere.

Por ejemplo, considere el circuito lógico en la figura 4-44(a). Supongamos que X y Y son ambos 1 y que Z está cambiando de 1 a 0. Entonces (b) muestra el diagrama de temporización, suponiendo que el retardo de propagación a través de cada compuerta o inversor sea de una unidad de tiempo. Aun cuando el análisis "estático" prediga que la salida es 1 para ambas combinaciones de entrada X,Y,Z = 111 y X,Y,Z = 110, el diagrama de temporización muestra que F tiende a 0 por una unidad de tiempo durante una transición 1-0 en Z, debido al retardo en el inversor que genera Z'.

Un *riesgo de 0 estático* es la posibilidad de una interferencia de 1 cuando esperamos que el circuito tenga una salida 0 estacionaria: *riesgo de 0 estático*

Definición: Un riesgo de 0 estático es un par de combinaciones de entrada que: a) difieren en solamente una variable de entrada y b) ambas proporcionan una salida 0; tal que es posible que ocurra una salida 1 momentánea durante una transición en la variable de entrada que difiere.

Puesto que un riesgo de 0 estático es justo el dual de un riesgo de 1 estático, un circuito OR-AND que es el dual de la figura 4-44(a) tendría un riesgo de 0 estático.

Un circuito OR-AND con cuatro riesgos de 0 estático se muestra en la figura 4-45(a). Uno de los riesgos ocurre cuando W,X,Y = 000 y Z está cambiando, como se ilustra en (b). Usted será capaz de encontrar los otros tres riesgos y eliminarlos todos después de estudiar la subsección siguiente.

*4.5.2 Hallando riesgos estáticos mediante el uso de mapas

Un mapa de Karnaugh puede utilizarse para detectar riesgos estáticos en un circuito de dos niveles de producto de sumas. La existencia o la inexistencia de riesgos estáticos depende del diseño del circuito para una función lógica.

Un circuito de producto de sumas (AND-OR) de dos niveles no tiene riesgos de 0 estático. Solamente existiría un riesgo de 0 estático en un circuito si tanto una variable como su complemento estuvieran conectados a la misma compuerta AND, lo que sería

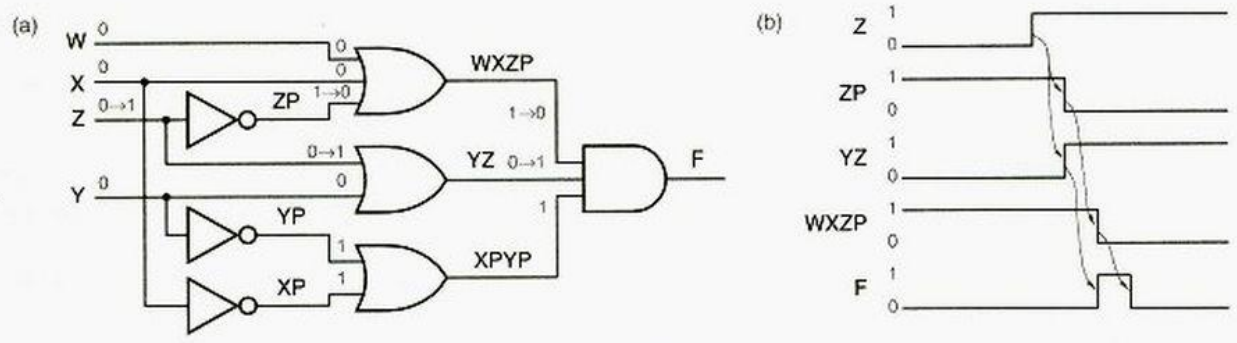


Figura 4-45 Circuito con riesgos de 0 estático: a) diagrama lógico; b) diagrama de temporización.

tonto. Sin embargo, el circuito *puede* tener riesgos de 1 estático. Su existencia puede predecirse a partir de un mapa de Karnaugh donde se marquen los términos de producto correspondientes a las compuertas AND en el circuito.

La figura 4-46(a) muestra el mapa de Karnaugh para el circuito de la figura 4-44. Es claro del mapa que no hay un término de producto simple que cubra ambas combinaciones de entrada X,Y,Z = 111 y X,Y,Z = 110. Así, de manera intuitiva, es posible para la salida "interferir" momentáneamente a 0 si la salida de la compuerta AND que cubre una de las combinaciones tiende a 0 antes de que la salida de la compuerta AND que cubre la otra combinación de entrada tienda a 1. La forma de eliminar el riesgo es también bastante aparente: simplemente incluya un término de producto extra (compuerta AND) para cubrir el par de entradas riesgosas, como se muestra en la figura 4-46(b). El término de producto extra, es el *consenso* de los dos términos originales; en general, debemos agregar términos de consenso para eliminar los riesgos. El circuito libre de riesgos correspondiente se ilustra en la figura 4-47.

consenso

Otro ejemplo se ilustra en la figura 4-48. En este ejemplo, tres términos de producto deben agregarse para eliminar los riesgos de 1 estático.

Un circuito de dos niveles (OR-AND) de producto de sumas apropiadamente diseñado no tiene riesgos de 1 estático. No obstante, *puede* tener riesgos de 0 estático. Estos riesgos pueden ser detectados y eliminados al estudiar los ceros adyacentes en el mapa de Karnaugh, de una manera dual para el precedente.

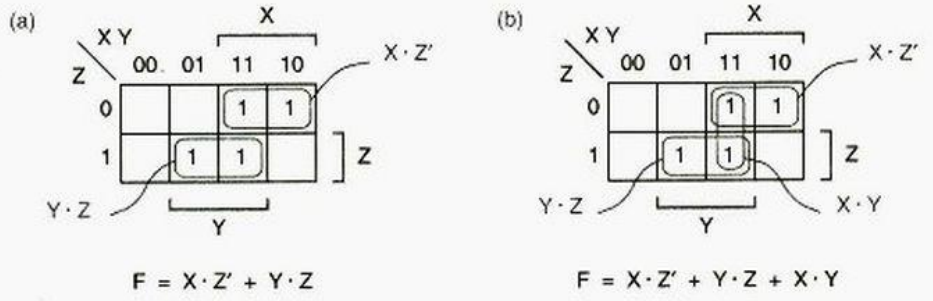


Figura 4-46
Mapa de Karnaugh para el circuito de la figura 4-44: a) como estaba originalmente diseñado; b) con el riesgo de 1 estático eliminado.

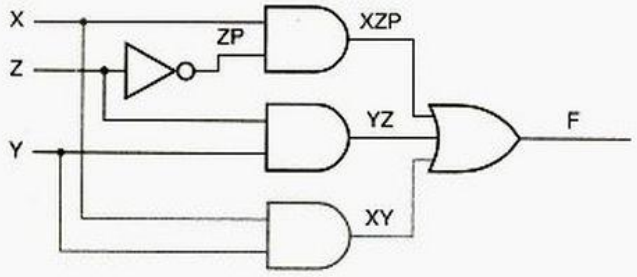


Figura 4-47
Circuito con el riesgo de 1 estático eliminado.

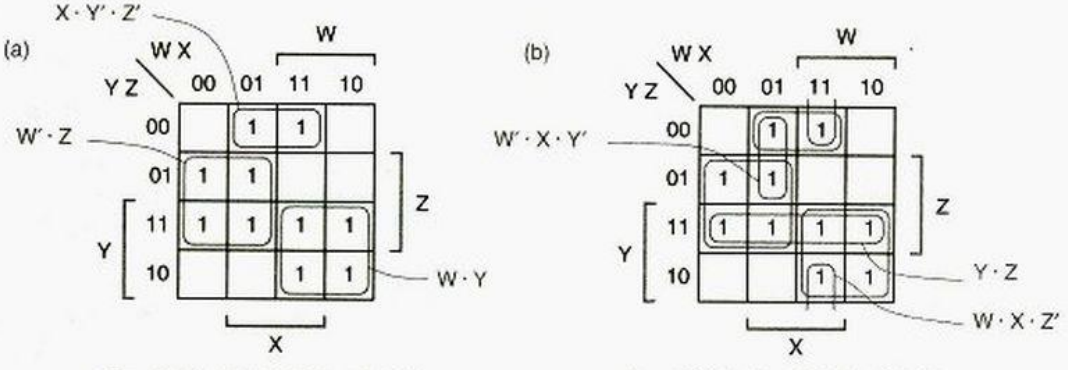
*4.5.3 Riesgos dinámicos

Un *riesgo dinámico* es la posibilidad de un cambio en la salida más de una vez como resultado de una transición de entrada simple. Las transiciones de salida múltiples pueden presentarse si hay múltiples trayectorias con diferentes retardos desde la entrada cambiante hasta la salida cambiante. *riesgo dinámico*

Por ejemplo, considere el circuito en la figura 4-49; tiene tres diferentes trayectorias desde la entrada X hasta la salida F. Una de las trayectorias va a través de una compuerta OR lenta, y otra va a través de una compuerta OR que es aún más lenta. Si la entrada al circuito es W,X,Y,Z = 0,0,0,1, entonces la salida será 1, como se muestra. Ahora supongamos que cambiamos la entrada X a 1. Suponiendo que todas las compuertas excepto las dos marcadas como "lenta" y "más lenta" sean muy rápidas, las transiciones mostradas en negro ocurren a continuación, y la salida tiende a 0. Con el tiempo, la salida de la compuerta OR "lenta" cambia, creando las transiciones mostradas en negritas cursivas, y la salida tiende a 1. Finalmente, la salida de la compuerta OR "más lenta" cambia, creando las transiciones mostradas en negritas cursivas, y la salida tiende a su estado final de 0.

Los riesgos dinámicos no se presentan en un circuito AND-OR u OR-AND de dos niveles apropiadamente diseñado, es decir, uno en el cual ninguna variable ni su complemento están conectados a la misma compuerta de primer nivel. En los circuitos de nivel

Figura 4-48 Mapa de Karnaugh para otro circuito de suma de productos: a) como se diseñó originalmente; b) con términos de producto extra para cubrir los riesgos de 1 estático.



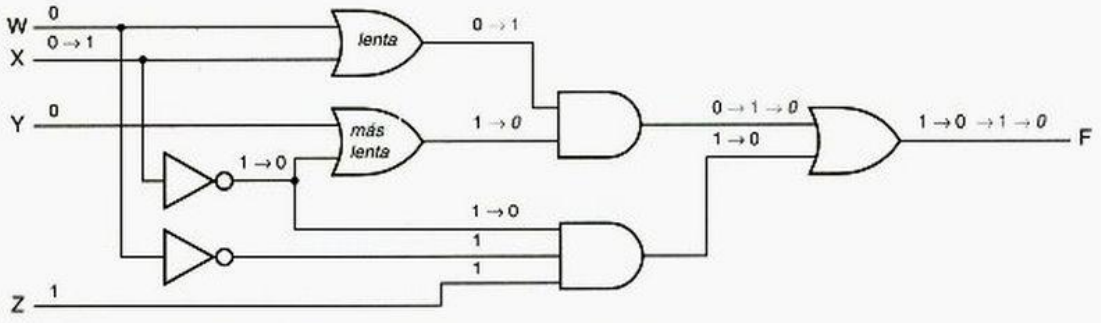


Figura 4-49 Circuito con un riesgo dinámico.

múltiple, los riesgos dinámicos pueden ser descubiertos utilizando un método descrito en las Referencias.

*4.5.4 Diseño de circuitos libres de riesgos

Solamente en algunas ocasiones, tales como el diseño de circuitos secuenciales de retroalimentación, se requiere de circuitos combinacionales libres de riesgos. Las técnicas para encontrar riesgos en circuitos arbitrarios, descritas en las Referencias, son bastante difíciles de utilizar. Así, cuando requiera un diseño libre de riesgos, será mejor emplear una estructura de circuito que sea fácil de analizar.

En particular, hemos indicado que un circuito AND-OR de dos niveles apropiadamente diseñado no tiene riesgos dinámicos o 0 estático. Los riesgos de 1 estático pueden existir en un circuito así, pero pueden ser encontrados y eliminados utilizando el método de mapa descrito anteriormente. Si el costo no es problema, entonces un método de fuerza bruta para obtener una realización libre de riesgos es emplear la suma completa: la suma de todos los implicantes primos de la función lógica (véase el ejercicio 4.84). De manera dual, un circuito OR-AND de dos niveles libre de riesgos puede diseñarse para cualquier función lógica. Finalmente, note que todo lo que hemos dicho acerca de los circuitos AND-OR se aplica naturalmente a los correspondientes diseños NAND-NAND, y lo que respecta a OR-AND se aplica a los NOR-NOR.

¡LA MAYORÍA DE LOS RIESGOS NO SON PELIGROSOS!

Cualquier circuito combinacional puede ser analizado en busca de riesgos. Sin embargo, un sistema digital *sincrónico*, bien diseñado está estructurado de modo que el análisis de riesgos no sea necesario para la mayoría de sus circuitos. En un sistema sincrónico, todas las entradas a un circuito combinacional se cambian a un tiempo particular y las salidas no se "examinan" hasta que han tenido tiempo para establecerse a un valor de estado estacionario. El análisis y eliminación de los riesgos son por lo regular necesarios solamente en el diseño de circuitos secuenciales asincrónicos, tales como los circuitos secuenciales de retroalimentación discutidos en la sección 7.9. Usted rara vez tendrá una razón para diseñar un circuito de esta clase, pero si lo hace, comprender de los riesgos será absolutamente esencial para obtener un resultado confiable.

4.6 El lenguaje de descripción de hardware ABEL

ABEL es un lenguaje de descripción de hardware (HDL, Hardware Description Language) que fue inventado para permitir a los diseñadores especificar funciones lógicas para su realización en los PLD. Un programa ABEL es un archivo de texto que contiene varios elementos:

- Documentación, incluyendo el nombre del programa y comentarios.
- Declaraciones que identifican las entradas y salidas de las funciones lógicas que serán efectuadas.
- Instrucciones que especifican las funciones lógicas que serán efectuadas.
- Por lo regular, una declaración del tipo de PLD u otro dispositivo destino en el que las funciones lógicas especificadas se realizarán.
- Generalmente, "vectores de prueba" que especifican las salidas esperadas de las funciones lógicas para ciertas entradas.

procesador de lenguaje ABEL

compilador ABEL

ABEL se apoya en un *procesador de lenguaje de ABEL*, el cual llamaremos simplemente un *compilador de ABEL*. El trabajo del compilador es traducir el archivo de texto de ABEL a un "patrón de fusión" que puede descargarse en un PLD físico. Aun cuando la mayoría de los PLD pueden ser físicamente programados solamente con patrones correspondientes a expresiones de suma de productos, ABEL permite que las funciones PLD sean expresadas con tablas de verdad o instrucciones `"IF"` anidadas, además de cualquier formato de expresión algebraica. El compilador manipula estos formatos y minimiza las ecuaciones resultantes para ajustar, si es posible, dentro de la estructura PLD disponible.

Hablaremos acerca de estructura PLD, patrones de fusión y temas relacionados posteriormente, en la sección 5.3, y mostraremos cómo dirigir programas ABEL para PLD específicos. En el medio tiempo, mostraremos cómo ABEL puede ser utilizado para especificar funciones lógicas combinacionales sin tener necesariamente que declarar el tipo de dispositivo fijado. Más adelante, en la sección 7.11, haremos lo mismo para las funciones lógicas secuenciales.

4.6.1 Estructura del programa ABEL

La tabla 4-10 muestra la estructura típica de un programa ABEL, y la tabla 4-11 muestra un programa real que presenta las características de lenguaje siguientes:

identificador

- *Identificadores*, deben comenzar con una letra o guión bajo, pueden contener hasta 31 letras, dígitos y guiones bajos, y distinguen entre mayúsculas y minúsculas.

`module` *(módulo)*

- Un archivo de programa comienza con una instrucción `module`, que asocia un identificador (`Alarm_Circuit`) con el módulo del programa. Los programas grandes pueden tener múltiples módulos, cada uno con su propio título local, declaraciones y ecuaciones. Note que en las palabras clave como "`module`" no hay distinción entre mayúsculas y minúsculas.

AVISO LEGAL ABEL (Advanced Boolean Equation Language) es una marca registrada de Data I/O Corporation (Redmond, WA 98073).

Tabla 4-10
Estructura típica de un programa ABEL.

```
module module name
title string
deviceID device deviceType;
pin declarations
other declarations
equations
equations
test_vectors
test vectors
end module name
```

title("título")

- La instrucción `title` especifica una cadena de título que se insertará en los archivos de documentación que son creados por el compilador.

cadena

- Una *cadena* es una serie de caracteres encerrados entre comillas simples.

device("dispositivo")

- La declaración opcional `device` incluye un identificador de dispositivo (`ALARMCKT`) y una cadena que denota el tipo de dispositivo (`'P16V8C'` para un GAL16V8). El compilador utiliza el identificador de dispositivo en los nombres de los archivos de documentación que genera, y emplea el tipo de dispositivo para determinar si el dispositivo puede en realidad efectuar las funciones lógicas especificadas en el programa.

comentario

- Los *comentarios* comienzan con una comilla doble y finalizan con otra comilla doble o con el fin de línea, cualquiera que se encuentre primero.

declaraciones de las terminales

- Las *declaraciones de terminales* le dicen al compilador acerca de los nombres simbólicos asociados con las terminales externas del dispositivo. Si el nombre de la señal es precedido con el prefijo NOT (!), entonces el complemento de la señal nombrada aparecerá en la terminal. Las declaraciones de terminales pueden o no incluir números de terminal; si no se da ninguno, el compilador los asigna con base en las capacidades del dispositivo señalado.

istype (tipo de señal de salida)
com ("combinacional")

- La palabra clave `istype` precede una lista de una o más propiedades, separadas por comas. Esto le dice al compilador el tipo de señal de salida. La palabra clave "com" indica una salida combinacional. Si no se proporciona una palabra clave `istype`, el compilador por lo general supone que la señal es una entrada a menos que aparezca en el lado izquierdo de una ecuación, en cuyo caso intenta averiguar las propiedades de salida del contexto. ¡Por su propia protección, es mejor utilizar precisamente la palabra clave `istype` para todas las salidas!

otras declaraciones

- *Otras declaraciones* permiten al diseñador definir constantes y expresiones para mejorar la legibilidad del programa y para simplificar el diseño lógico.

equations ("ecuaciones")

- La instrucción `equations` indica que las ecuaciones lógicas que definen las señales de salida como funciones de señales de entrada siguen a continuación.

ecuaciones

- Las *ecuaciones* se escriben como instrucciones de asignación en un lenguaje de programación convencional. Cada ecuación es terminada por un punto y coma. ABEL utiliza los símbolos siguientes para operaciones lógicas:

Tabla 4-11 Un programa ABEL para el circuito de alarma de la figura 4-19.

```
module Alarm_Circuit
title 'Alarm Circuit Example
J. Wakerly, Micro Systems Engineering'
ALARMCKT device 'P16V8C';

' Input pins
PANIC, ENABLEA, EXITING      pin 1, 2, 3;
WINDOW, DOOR, GARAGE         pin 4, 5, 6;
' Output pins
ALARM                        pin 11 istype 'com';

' Constant definition
X = .X.;

" Intermediate equation
SECURE = WINDOW & DOOR & GARAGE;

equations
ALARM = PANIC # ENABLEA & !EXITING & !SECURE;

test_vectors
([PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE] -> [ALARM])
[    1,    .X.,    .X.,   .X., .X.,   .X.] -> [    1];
[    0,      0,    .X.,   .X., .X.,   .X.] -> [    0];
[    0,      1,      1,   .X., .X.,   .X.] -> [    0];
[    0,      1,      0,     0, .X.,   .X.] -> [    1];
[    0,      1,      0,   .X.,   0,   .X.] -> [    1];
[    0,      1,      0,   .X., .X.,     0] -> [    1];
[    0,      1,      0,     1,   1,     1] -> [    0];

end Alarm_Circuit
```

&	AND.	*& (AND)*
#	OR.	*# (OR)*
!	NOT (usado como prefijo).	*! (NOT)*
$	XOR.	*$ (XOR)*
!$	XNOR.	*!$ (XNOR)*

Como en los lenguajes de programación convencionales, AND (&) tiene precedencia sobre OR (#) en las expresiones. La directiva `@ALTERNATE` hace que el compilador reconozca un conjunto alterno de símbolos para estas operaciones: `+`, `*`, `/`, `:+:` y `:*:`, respectivamente. Este libro hace uso de los símbolos predeterminados en todo el mismo. *@ALTERNATE*

- La instrucción opcional `test_vectors` indica que a continuación vienen los vectores de prueba. `test_vectors`
- Los *vectores de prueba* asocian las combinaciones de entrada con los valores esperados de salida; se utilizan para simulación y prueba como se explica en la sección 4.6.7. *vectores de prueba*

.X. ("sin importancia")

- El compilador reconoce varias constantes especiales, incluyendo .X., un bit simple cuyo valor es "sin importancia".

end ("final")

- La instrucción *end* marca el final del módulo.

operador de asignación no temporizado, =

Las ecuaciones para salidas combinacionales utilizan el *operador de asignación no temporizado*, =. el lado izquierdo de la ecuación normalmente contiene un nombre de señal. El lado derecho es una expresión lógica, no necesariamente en la forma de producto de sumas. El nombre de la señal en el lado izquierdo de una ecuación puede ser opcionalmente precedido por el operador NOT, !; esto es equivalente a la complementación del lado derecho. El trabajo del compilador es generar un patrón de fusión de tal modo que la señal nombrada en el lado izquierdo realice la expresión lógica en el lado derecho.

4.6.2 Funcionamiento del compilador de ABEL

El programa en la tabla 4-11 realiza la función de alarma que describimos desde la página 215. La señal llamada ENABLE ha sido codificada como ENABLEA porque en ABEL la palabra ENABLE es una palabra reservada.

ecuación intermedia

Note que no todas las ecuaciones aparecen bajo la instrucción equations. Una ecuación intermedia para un identificador SEGURO aparece con anterioridad. Esta ecuación es meramente una definición que asocia una expresión con el identificador SEGURO. El compilador ABEL sustituye esta expresión para el identificador SEGURO en cada lugar en que aparece SEGURO después de su definición.

En la figura 4-19 de la página 216 realizamos el circuito de alarma directamente de las expresiones SEGURO y ALARMA, utilizando múltiples niveles de lógica. El compilador ABEL no emplea expresiones para interconectar compuertas en esta manera. en vez de ello, "trunca" las expresiones para obtener un resultado de suma de productos de dos niveles mínimo apropiado para la realización en un PLD. De este modo, cuando compila, la tabla 4-11 produciría un resultado equivalente al circuito AND-OR que mostramos en la figura 4-20 de la página 216, la cual parece ser mínima.

De hecho, lo es. La tabla 4-12 muestra el archivo de ecuaciones sintetizadas creado por el compilador ABEL. Note que el compilador crea ecuaciones solamente para la señal ALARMA, la única salida. La señal SEGURO no aparece por ningún lado.

El compilador encuentra una expresión mínima de suma de productos tanto para ALARMA como para su complemento, !ALARMA. Como se mencionó con anterioridad, muchos PLD tienen la capacidad selectiva de invertir, para invertir o no sus salidas AND-OR. La "ecuación de polaridad inversa" en la tabla 4-12 es una realización de suma de productos de !ALARMA y sería utilizada si fuera seleccionada la inversión de salida.

En este ejemplo, la ecuación de polaridad inversa tiene un término de producto menos que la ecuación de polaridad normal para ALARMA, de modo que el compilador seleccionará esta ecuación si el dispositivo señalado tiene inversión de salida seleccionable. Un usuario también puede forzar al compilador para utilizar ya sea la polaridad normal o inversa para una señal al incluir la palabra clave "buffer" o "invert", respectivamente, en la lista de propiedades istype de la señal. (Con algunos compiladores de ABEL, las palabras clave "pos" y "neg" pueden emplearse para este propósito, pero vea la sección 4.6.6.)

Tabla 4-12 Archivo de ecuaciones sintetizadas producido por ABEL para el programa en la tabla 4-11.

```
ABEL 6.30

Design alarmckt created Tue Nov 24 1998

Title: Alarm Circuit Example
Title: J. Wakerly, Micro Systems Engineering

 P-Terms   Fan-in  Fan-out  Type  Name (attributes)
---------  ------  -------  ----  ----------------
   4/3        6        1    Pin   ALARM
=========
   4/3           Best P-Term Total: 3
                       Total Pins: 7
                      Total Nodes: 0
              Average P-Term/Output: 3


Equations:

ALARM = (ENABLEA & !EXITING & !DOOR
     # ENABLEA & !EXITING & !WINDOW
     # ENABLEA & !EXITING & !GARAGE
     # PANIC);


Reverse-Polarity Equations:

!ALARM = (!PANIC & WINDOW & DOOR & GARAGE
     # !PANIC & EXITING
     # !PANIC & !ENABLEA);
```

4.6.3 Instrucciones WHEN y bloques de ecuaciones

instrucción WHEN

Además de las ecuaciones, ABEL proporciona la *instrucción WHEN* como otro medio para especificar funciones de lógica combinacional en la sección *equations* de un programa ABEL. La tabla 4-13 muestra la estructura general de una instrucción WHEN, similar a una instrucción IF en un lenguaje de programación convencional. La cláusula ELSE es opcional. Aquí *ExpresionLogica* es una expresión que resulta en un valor de verdadero (1) o falso (0). Que se ejecute la *EcuaciónVerdadera* o *EcuaciónFalsa*

```
WHEN LogicExpression THEN
   TrueEquation;
ELSE
   FalseEquation;
```

Tabla 4-13 Estructura de una instrucción WHEN de ABEL.

depende del valor de *ExpresiónLógica*. Pero necesitamos ser un poco más precisos acerca de lo que queremos decir por "ejecutadas", como se discute a continuación.

En el caso más simple, *EcuaciónVerdadera* y la opcional *EcuaciónFalsa* son instrucciones de asignación, como en las primeras dos instrucciones `WHEN` en la tabla 4-14 (para `X1` y `X2`). En este caso, *ExpresiónLógica* es operada lógicamente (**AND**) con el lado derecho de *EcuaciónVerdadera* y el complemento de *ExpresiónLógica* se opera con **AND** con el lado derecho de *EcuaciónFalsa*. De este modo, las ecuaciones para `X1A` y `X2A` producen los mismos resultados que las correspondientes instrucciones `WHEN` pero sin emplear `WHEN`.

Note en el primer ejemplo que `X1` aparece en la *EcuaciónVerdadera*, pero no hay *EcuaciónFalsa*. Así, ¿qué ocurre a `X1` cuando *ExpresiónLógica* (`!A#B`) es falsa? Se puede pensar que el valor de `X1` debería ser sin importancia para estas combinaciones de entrada, pero no lo es, como se explica a continuación.

Formalmente, el operador de asignación no temporizado, =, especifica las combinaciones de entrada que deberían agregarse al conjunto de iniciación para la señal de salida que aparece en el lado izquierdo de la ecuación. Un conjunto de iniciación de la salida empieza vacío y se aumenta cada vez que la salida aparece en el lado izquierdo de una ecuación. Es decir, los lados derechos de todas las ecuaciones para la misma salida (no complementada) se operan con **OR** juntos. (Si la salida aparece complementada en el lado izquierdo, el lado derecho se complementa antes de ser operado con ORed). Así, el valor de `X1` es 1 solamente para las combinaciones de entrada para las cuales *ExpresiónLógica* (`!A#B`) sea verdad y el lado derecho de *EcuaciónVerdadera* (`C&!D`) sea también verdadero.

En el segundo ejemplo, `X2` aparece en el lado izquierdo de las dos ecuaciones, de modo que la ecuación equivalente mostrada para `X2A` se obtiene al operar con **OR** los dos lados derechos después de hacerlo con **AND** para cada uno con la condición apropiada.

La *EcuaciónVerdadera* y la opcional *EcuaciónFalsa* en una instrucción `WHEN` pueden ser cualquier ecuación. Además, las instrucciones `WHEN` pueden estar "anidadas" al utilizar otra instrucción `WHEN` como la *EcuaciónFalsa*. Cuando las instrucciones se encuentran anidadas, todas las condiciones que conducen a una instrucción "ejecutada" están operadas con **AND**. La ecuación para `X3` y su homólogo menos `WHEN` para `X3A` en la tabla 4-14 ilustran el concepto.

La *EcuaciónVerdadera* puede ser otra instrucción `WHEN` si se encuentra encerrada entre signos de "llaves" ({ }), como se muestra en el ejemplo `X4` en la tabla. Esto es apenas un caso del uso general de las llaves descrito brevemente.

Aunque todos nuestros ejemplos `WHEN` han asignado valores a la misma salida dentro de cada parte de una instrucción `WHEN` dada, este no tiene que ser el caso. La instrucción `WHEN` del segundo caso al último en la tabla 4-14 es un ejemplo de ello.

Con frecuencia es útil hacer más de una asignación en *EcuaciónVerdadera* o *EcuaciónFalsa* o ambas. Para este propósito, ABEL respalda los bloques de ecuación en cualquier sitio en que respalde una ecuación simple. Un *bloque de ecuación* es sencillamente una serie de instrucciones encerradas entre llaves, como se muestra en la última instrucción `WHEN` en la tabla. Las instrucciones individuales en la serie pueden ser instrucciones de asignación simples o pueden ser instrucciones `WHEN` o bloques de ecuación anidados. No se usa un punto y coma después de una llave de cierre de bloque. Sólo por diversión, la tabla 4-15 muestra las ecuaciones que el compilador ABEL produce para todo el programa de ejemplo.

bloque de ecuación

Tabla 4-14 Ejemplos de instrucciones WHEN.

```
module WhenEx
title 'WHEN Statement Examples'

' Input pins
A, B, C, D, E, F                pin;

' Output pins
X1, X1A, X2, X2A, X3, X3A, X4   pin istype 'com';
X5, X6, X7, X8, X9, X10         pin istype 'com';

equations

WHEN (!A # B) THEN X1 = C & !D;

X1A = (!A # B) & (C & !D);

WHEN (A & B) THEN X2 = C # D;
ELSE X2 = E # F;

X2A = (A & B) & (C # D)
    # !(A & B) & (E # F);

WHEN (A) THEN X3 = D;
ELSE WHEN (B) THEN X3 = E;
ELSE WHEN (C) THEN X3 = F;

X3A = (A) & (D)
    # !(A) & (B) & (E)
    # !(A) & !(B) & (C) & (F);

WHEN (A) THEN
  {WHEN (B) THEN X4 = D;}
ELSE X4 = E;

WHEN (A & B) THEN X5 = D;
ELSE WHEN (A # !C) THEN X6 = E;
ELSE WHEN (B # C) THEN X7 = F;

WHEN (A) THEN {
    X8 = D & E & F;
    WHEN (B) THEN X8 = 1; ELSE {X9 = D; X10 = E;}
} ELSE {
    X8 = !D # !E;
    WHEN (D) THEN X9 = 1;
    {X10 = C & D;}
}

end WhenEx
```

Tabla 4-15 Archivo de ecuaciones sintetizadas producido por ABEL para el programa en la tabla 4-14.

```
ABEL 6.30

Design whenex created Wed Dec 2 1998

Title: WHEN Statement Examples

 P-Terms  Fan-in Fan-out Type Name
--------- ------ ------- ---- -----
   2/3       4       1   Pin  X1
   2/3       4       1   Pin  X1A
   6/3       6       1   Pin  X2
   6/3       6       1   Pin  X2A
   3/4       6       1   Pin  X3
   3/4       6       1   Pin  X3A
   2/3       4       1   Pin  X4
   1/3       3       1   Pin  X5
   2/3       4       1   Pin  X6
   1/3       3       1   Pin  X7
   4/4       5       1   Pin  X8
   2/2       3       1   Pin  X9
   2/4       5       1   Pin  X10
=========
  36/42      Best P-Term Total: 30
                   Total Pins: 19
                  Total Nodes: 0
        Average P-Term/Output: 2
```

```
Equations:

X1 = (C & !D & !A
     # C & !D & B);

X1A = (C & !D & !A
     # C & !D & B);

X2 = (D & A & B
     # C & A & B
     # !B & E
     # !A & E
     # !B & F
     # !A & F);

X2A = (D & A & B
     # C & A & B
     # !B & E
     # !A & E
     # !B & F
     # !A & F);

X3 = (C & !A & !B & F
     # !A & B & E
     # D & A);

X3A = (C & !A & !B & F
     # !A & B & E
     # D & A);

X4 = (D & A & B
     # !A & E);

X5 = (D & A & B);

X6 = (A & !B & E
     # !C & !A & E);

X7 = (C & !A & F);

X8 = (D & A & E & F
     # A & B
     # !A & !E
     # !D & !A);

X9 = (D & !A
     # D & !B);

X10 = (C & D & !A
     # A & !B & E);
```

```
Reverse-Polarity Eqns:

!X1 = (A & !B
     # D
     # !C);

!X1A = (A & !B
     # D
     # !C);

!X2 = (!C & !D & A & B
     # !B & !E & !F
     # !A & !E & !F);

!X2A = (!C & !D & A & B
     # !B & !E & !F
     # !A & !E & !F);

!X3 = (!C & !A & !B
     # !A & B & !E
     # !D & A
     # !A & !B & !F);

!X3A = (!C & !A & !B
     # !A & B & !E
     # !D & A
     # !A & !B & !F);

!X4 = (A & !B
     # !D & A
     # !A & !E);

!X5 = (!A
     # !D
     # !B);

!X6 = (A & B
     # C & !A
     # !E);

!X7 = (A
     # !C
     # !F);

!X8 = (A & !B & !F
     # D & !A & E
     # A & !B & !E
     # !D & A & !B);

!X9 = (!D
     # A & B);

!X10 = (A & B
     # !D & !A
     # !C & !A
     # A & !E);
```

```
truth_table (input-list -> output-list)
            input-value -> output-value;
            ...
            input-value -> output-value;
```

Tabla 4-16
Estructura de una tabla de verdad de ABEL.

4.6.4 Tablas de verdad

ABEL proporciona una forma más para especificar funciones lógicas combinacionales: la *tabla de verdad*, con el formato general mostrado en la tabla 4-16. La palabra clave `truth_table` introduce una tabla de verdad. Las *lista-entrada* y *lista-salida* suministran los nombres de las señales de entrada y las salidas que ellas afectan. Cada una de estas listas es o un nombre de señal simple o un *conjunto*; los conjuntos se describen completamente en la sección 4.6.5. A continuación de la introducción de la tabla de verdad se encuentran una serie de instrucciones, cada una de las cuales especifica un valor de entrada y un valor de salida requerido utilizando el operador "->". Por ejemplo, la tabla de verdad para un inversor se muestra a continuación:

tabla de verdad
`truth_table`
lista-entrada
lista-salida
operador de tabla de verdad no temporizado, ->

```
truth_table (X -> NOTX)
            0 -> 1;
            1 -> 0;
```

No es necesario que la lista de valores de entrada esté completa; solamente el conjunto de iniciación de la función necesita especificarse a menos que se encuentre habilitado el procesamiento de valores sin importancia (véase la sección 4.6.6). La tabla 4-17 muestra cómo la función del detector de números primos descrita en la página 215 puede especificarse empleando un programa ABEL. Por comodidad, el identificador `NUM` se define como un sinónimo para el conjunto de cuatro bits de entrada `[N3,N2,N1,N0]`, permitiendo que se escriba un valor de entrada de 4 bits como un entero decimal.

Tabla 4-17 Un programa ABEL para el detector de números primos.

```
module PrimeDet
title '4-Bit Prime Number Detector'

" Input and output pins
N0, N1, N2, N3                  pin;
F                               pin istype 'com';

" Definition
NUM = [N3,N2,N1,N0];

truth_table (NUM -> F)
              1 -> 1;
              2 -> 1;
              3 -> 1;
              5 -> 1;
              7 -> 1;
             11 -> 1;
             13 -> 1;
end PrimeDet
```

Tanto las tablas de verdad como las ecuaciones pueden utilizarse dentro del mismo programa ABEL. La palabra clave `equations` introduce una serie de ecuaciones, mientras que la palabra clave `truth_table` introduce una sola tabla de verdad.

4.6.5 Intervalos, conjuntos y relaciones

La mayoría de los sistemas incluyen buses, registros y otros circuitos que manejan un grupo de dos o más señales de manera idéntica. ABEL proporciona varios atajos para la definición y el uso convenientes de tales señales.

intervalo

El primer atajo es para nombrar señales numeradas similares. Como se muestra en las definiciones de terminales en la tabla 4-18, un *intervalo* de nombres de señal puede definirse al establecer el primero y el último nombres del intervalo, separados por "..". Por ejemplo, escribir "`N3..N0`" es lo mismo que escribir "`N3,N2,N1,N0`". Note en la tabla que el intervalo puede ser ascendiente o descendiente.

conjunto

A continuación, necesitamos una facilidad para escribir ecuaciones de manera más compacta cuando un grupo de señales se manejan todas de manera idéntica, a fin de reducir la posibilidad de errores e inconsistencias. Un *conjunto* de ABEL es sencillamente una colección de señales que se manejan como una unidad. Cuando se aplica una operación lógica como AND, OR o bien cuando se aplica una asignación a un conjunto, es aplicada a cada elemento de dicho conjunto.

Cada conjunto se define al principio del programa al asociar un nombre de conjunto con una lista entre corchetes de los elementos del conjunto (por ejemplo, `N=[N3,N2,N1,N0]` en la tabla 4-18). La lista de elementos del conjunto puede emplear notación abreviada (`YOUT=[Y1..Y4]`), pero los nombres de elemento necesitan no ser semejantes o tener cualquier correspondencia con el nombre del conjunto (`COMP=[EQ,GE]`). Los elementos del conjunto también pueden ser constantes (`GT=[0,1]`). En cualquier caso, el número y orden de los elementos en un conjunto son significativos, como veremos más adelante.

La mayoría de los operadores de ABEL pueden aplicarse a conjuntos. Cuando una operación se aplica a dos o más conjuntos, todos éstos deben tener el mismo número de elementos, y la operación se aplica de manera individual a los elementos del conjunto en posiciones parecidas, sin importar sus nombres o números. De este modo, la ecuación "`YOUT = N & M`" es equivalente a cuatro ecuaciones:

```
Y1 = N3 & M3;
Y2 = N2 & M2;
Y3 = N1 & M1;
Y4 = N0 & M0;
```

Cuando una operación incluye tanto variables de conjunto como no pertenecientes a un conjunto, las variables no pertenecientes al conjunto se combinan de manera individual con los elementos del conjunto en cada posición. De esta forma, la ecuación "`ZOUT = (SEL & N) # (!SEL & M)`" es equivalente a cuatro ecuaciones de la forma "`Zi = (SEL & Ni) # (!SEL & Mi)`" para `i` igual de `0` a `3`.

relación
operador relacional

Otra importante característica es la capacidad de ABEL para convertir "relaciones" en expresiones lógicas. Una *relación* es un par de operandos combinados con uno de los *operadores relacionales* enumerados en la tabla 4-19. El compilador convierte una relación en una expresión lógica que es 1 si y sólo si la relación es verdadera.

Los operandos en una relación son tratados como enteros sin signo, y el operando puede ser ya sea un entero o un conjunto. Si el operando es un conjunto, es tratado como un

Tabla 4-18 Ejemplo de intervalos, conjuntos y relaciones en ABEL.

```
module SetOps
title 'Set Operation Examples'

" Input and output pins
N3..N0, M3..M0, SEL                          pin;
Y1..Y4, Z0..Z3, EQ, GE, GTR, LTH, UNLUCKY    pin istype 'com';

" Definitions
N    = [N3,N2,N1,N0];
M    = [M3,M2,M1,M0];
YOUT = [Y1..Y4];
ZOUT = [Z3..Z0];

COMP = [EQ,GE];
GT   = [ 0, 1];
LT   = [ 0, 0];

equations

YOUT = N & M;
ZOUT = (SEL & N) # (!SEL & M);
EQ = (N == M);
GE = (N >= M);
GTR = (COMP == GT);
LTH = (COMP == LT);
UNLUCKY = (N == 13) # (M == ^hD) # ((N + M) == ^b1101);

end SetOps
```

entero binario sin signo con la variable más a la izquierda representando el bit más significativo. De manera predeterminada, se supone que los números en los programas ABEL son de base 10. Los números hexadecimales y binarios se denotan mediante un prefijo de "^h" o "^b", respectivamente, como se muestra en la última ecuación en la tabla 4-18.

prefijo hexadecimal ^h
prefijo binario ^b

Los conjuntos y relaciones en ABEL permiten mucha funcionalidad para expresarse en muy pocas líneas de código. Por ejemplo, las ecuaciones en la tabla 4-18 generan ecuaciones minimizadas con 69 términos de producto, como se muestra en el resumen de la tabla 4-20.

Tabla 4-19 Operadores relacionales en ABEL.

Símbolo	*Relación*
==	igual
!=	distinto de
<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que

Tabla 4-20 Resumen de ecuaciones sintetizadas producidas por ABEL para el programa en la tabla 4-18.

```
P-Terms   Fan-in  Fan-out  Type  Name (attributes)
---------  ------  -------  ----  ----------------
   1/2        2        1   Pin   Y1
   1/2        2        1   Pin   Y2
   1/2        2        1   Pin   Y3
   1/2        2        1   Pin   Y4
   2/2        3        1   Pin   Z0
   2/2        3        1   Pin   Z1
   2/2        3        1   Pin   Z2
   2/2        3        1   Pin   Z3
  16/8        8        1   Pin   EQ
  23/15       8        1   Pin   GE
   1/2        2        1   Pin   GTR
   1/2        2        1   Pin   LTH
  16/19       8        1   Pin   UNLUCKY
=========
  69/62          Best P-Term Total: 53
                       Total Pins: 22
                      Total Nodes: 0
               Average P-Term/Output: 4
```

*4.6.6 Entradas sin importancia

Algunas versiones del compilador de ABEL tienen una capacidad limitada para manipular entradas sin importancia. Como ya se mencionó, las ecuaciones ABEL especifican combinaciones de entrada que pertenecen al conjunto de iniciación de una función lógica; se supone que las combinaciones restantes pertenecen al conjunto de neutralización. Si algunas combinaciones de entrada pueden en su lugar asignarse al conjunto d, entonces el programa puede ser capaz de usar esas entradas sin importancia para hacer un mejor trabajo de minimización.

El lenguaje ABEL define dos mecanismos para asignación de combinaciones de entrada al conjunto d. A fin de utilizar cualquier mecanismo, se debe incluir la directiva de compilación `@DCSET` en su programa o incluir "`dc`" en la lista de propiedades `istype` de las salidas para las cuales usted desea que se tomen en cuenta las "sin importancia".

`@DCSET`

`dc`

`?=` operador de asignación no temporizado "sin importancia"

El primer mecanismo es el *operador de asignación no temporizado sin importancia*, `?=`. Este operador se emplea en lugar de = en ecuaciones para indicar que las combinaciones de entrada coincidan con el lado derecho deberían ser colocadas en el conjunto d en vez del conjunto de inicialización. Aunque este operador está documentado en el compilador ABEL que yo utilizo, desgraciadamente parece estar defectuoso, de modo que no voy a hablar más acerca de ello.

El segundo mecanismo es la tabla de verdad. Cuando se habilita el procesamiento de los "sin importancia", cualquiera de las combinaciones de entrada que no se encuentran explícitamente enumeradas en la tabla de verdad se colocan en el conjunto d. De este modo, el detector de dígitos BCD primos descrito en la página 232 puede ser especificado en ABEL como se muestra en la tabla 4-21. Un valor sin importancia es implicado para combinaciones de entrada 10-15 debido a que estas combinaciones no aparecen en la tabla de verdad y la directiva `@DCSET` está en efecto.

```
module DontCare
title 'Dont Care Examples'
@DCSET

" Input and output pins
N3..N0, A, B                    pin;
F, Y                            pin istype 'com';

NUM = [N3..N0];
X = .X.;

truth_table (NUM->F)
              0->0;
              1->1;
              2->1;
              3->1;
              4->0;
              5->1;
              6->0;
              7->1;
              8->0;
              9->0;

truth_table ([A,B]->Y)
             [0,0]->0;
             [0,1]->X;
             [1,0]->X;
             [1,1]->1;

end DontCare
```

Tabla 4-21
Programa ABEL que utiliza entradas "sin importancia".

También es posible especificar combinaciones sin importancia de manera explícita, como se muestra en la segunda tabla de verdad. Como se presentó muy al principio de esta sección, ABEL reconoce `.X.` como una constante especial de un bit cuyo valor es "sin importancia". En la tabla 4-21, el identificador "X" ha sido igualado a esta constante precisamente para hacerlo más fácil de escribir entradas sin importancia en la tabla de verdad. Las ecuaciones minimizadas resultantes de la tabla 4-21 se muestran en la tabla 4-22. Note que las dos ecuaciones para F no son iguales; el compilador ha seleccionado diferentes valores para las "sin importancia".

```
Equations:
F = (!N2 & N1
     # !N3 & N0);
Y = (B);

Reverse-Polarity Equations:
!F = (N2 & !N0
     # N3
     # !N1 & !N0);
!Y = (!B);
```

Tabla 4-22
Ecuaciones minimizadas derivadas de la tabla 4-21.

Tabla 4-23
Estructura de los vectores de prueba ABEL.

```
test_vectors  (input-list -> output-list)
              input-value -> output-value;
              ...
              input-value -> output-value;
```

4.6.7 Vectores de prueba

Los programas ABEL pueden contener vectores de prueba, como mostramos en la tabla 4-11 de la página 251. El formato general de los vectores de prueba es muy semejante al de una tabla de verdad y se muestra en la tabla 4-23. La palabra clave `test_vectors` introduce una tabla de verdad. La *lista-entrada* y la *lista-salida* proporcionan los nombres de las señales de entrada y las salidas que aquéllas afectan. Cada una de estas listas es ya sea un nombre de señal simple o un conjunto. A continuación de la introducción del vector de prueba tenemos una serie de instrucciones, cada una de las cuales especifica un valor de entrada y un valor de salida esperado al utilizar el operador "->".

`test_vectors`
lista-entrada
lista-salida

Los vectores de prueba ABEL tienen dos usos y propósitos principales:

1. Después que el compilador ABEL traduce el programa en "patrón de fusión" para un dispositivo en particular, simula la operación del dispositivo programado final al aplicar las entradas del vector de prueba a un modelo de software del dispositivo y comparar sus salidas con las correspondientes salidas del vector de prueba. El diseñador puede especificar una serie de vectores de prueba a fin de verificar doblemente que ese dispositivo se comportará como se espera para algunas o todas las combinaciones de entrada.
2. Después de que un PLD se programa físicamente, la unidad de programación aplica las entradas del vector de prueba al dispositivo físico y compara las salidas del dispositivo con las correspondientes salidas del vector de prueba. Esto se hace para verificar la correcta programación y funcionamiento del dispositivo.

Desgraciadamente, los vectores de prueba de ABEL rara vez hacen un muy buen trabajo en cualquiera de estas tareas, como explicaremos a continuación.

Los vectores de prueba de la tabla 4-11 se repiten en la tabla 4-24, excepto que para efectos de legibilidad supondremos que el identificador `X` ha sido igualado a la constante "sin importancia" `.X.`, y hemos agregado comentarios para numerar los vectores de prueba.

La tabla 4-24 en realidad parece ser un muy conveniente conjunto de vectores de prueba. Desde el punto de vista del diseñador, estos vectores cubren en su totalidad la operación esperada del circuito de alarma, como se detalla acto seguido, vector por vector:

1. Si `PANIC` (PÁNICO) es 1, entonces la salida de alarma (F) debería estar encendida sin tener en cuenta los otros valores de entrada. Todos los vectores restantes cubren los casos donde `PANIC` es 0.
2. Si la alarma no está habilitada, entonces la salida debería estar apagada.
3. Si la alarma se encuentra habilitada pero estamos saliendo, entonces la salida debería apagarse.

4-6. Si la alarma está habilitada y no hemos salido, entonces la salida debería estar encendida si cualquiera de las señales de sensor `WINDOW,DOOR` o `GARAGE` son 0.

7. Si la alarma se encuentra habilitada, no hemos salido y todas las señales de sensores son 1, entonces la salida debería estar apagada.

```
test_vectors
([PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE] -> [ALARM])
[   1,      X,      X,      X,    X,     X] -> [    1];  '1
[   0,      0,      X,      X,    X,     X] -> [    0];  '2
[   0,      1,      1,      X,    X,     X] -> [    0];  '3
[   0,      1,      0,      0,    X,     X] -> [    1];  '4
[   0,      1,      0,      X,    0,     X] -> [    1];  '5
[   0,      1,      0,      X,    X,     0] -> [    1];  '6
[   0,      1,      0,      1,    1,     1] -> [    0];  '7
```

Tabla 4-24
Vectores de prueba para el circuito de alarma en la tabla 4-11.

El problema es que ABEL no maneja las entradas sin importancia como debería hacerlo. Por ejemplo, el vector de prueba 1 debería probar 32 distintas combinaciones de entrada correspondientes a todas las 32 combinaciones de entradas sin importancia ENABLEA, EXITING, WINDOW, DOOR y GARAGE. Pero no lo hace. En esta situación, el compilador ABEL interpreta "sin importancia" como: "al usuario no le interesa cuál valor de entrada yo utilice" y asigna precisamente 0 a todas las entradas sin importancia en un vector de prueba. En este ejemplo, se podría haber escrito incorrectamente la ecuación de salida como "F = PANIC & !ENABLEA # ENABLEA & ..."; los vectores de prueba todavía pasarían, aun cuando el botón de pánico funcionaría solamente cuando el sistema se encuentre deshabilitado.

El segundo uso de los vectores de prueba se encuentra en la prueba del dispositivo físico. La mayoría de los defectos físicos en los dispositivos lógicos pueden detectarse utilizando el *modelo de bloqueo en falla simple*, que supone que cualquier defecto físico es equivalente a tener una entrada o salida de compuerta simple bloqueada en un valor de 0 o 1 lógico. Sólo colocar junto un conjunto de vectores de prueba que parezca ejercitar las especificaciones funcionales de un circuito, como lo hicimos en la tabla 4-24, no garantiza que todos los bloqueos en fallas simples sean detectados. Los vectores de prueba tienen que ser elegidos de tal modo que todo posible bloqueo en falla cause un valor incorrecto en la salida del circuito para alguna combinación de entrada de vector de prueba.

modelo de bloqueo en falla simple

La tabla 4-25 muestra un conjunto completo de vectores de prueba para el circuito de alarma cuando se realiza como un circuito de suma de productos de dos niveles. Los primeros cuatro vectores verifican bloqueos en 1 de fallas en la compuerta OR mientras que los últimos tres verifican bloqueos en 0 de fallas en las compuertas AND; resulta que esto es suficiente para detectar todos los bloqueos en fallas simples. Si se sabe algo acerca de la prueba de fallas se puede generar vectores de prueba para pequeños circuitos a mano (como lo hice en este ejemplo), pero la mayoría de los diseñadores utilizan herramientas automatizadas de terceras partes para crear vectores de prueba de alta calidad para sus diseños PLD.

```
test_vectors
([PANIC,ENABLEA,EXITING,WINDOW,DOOR,GARAGE] -> [ALARM])
[   1,      0,      1,      1,    1,     1] -> [    1];  '1
[   0,      1,      0,      0,    1,     1] -> [    1];  '2
[   0,      1,      0,      1,    0,     1] -> [    1];  '3
[   0,      1,      0,      1,    1,     0] -> [    1];  '4
[   0,      0,      0,      0,    0,     0] -> [    0];  '5
[   0,      1,      1,      0,    0,     0] -> [    0];  '6
[   0,      1,      0,      1,    1,     1] -> [    0];  '7
```

Tabla 4-25
Vectores de prueba de bloqueo en falla simple para la realización de suma de productos mínima del circuito de alarma.

4.7 El lenguaje de descripción de hardware VHDL

VHDL

A mediados de la década de 1980, el Departamento de Defensa de Estados Unidos (DoD, Department of Defense) y el IEEE patrocinaron el desarrollo de un lenguaje de descripción de hardware de alta capacidad denominado *VHDL*. El lenguaje arrancó con las siguientes características, que todavía tiene:

- Los diseños pueden descomponerse jerárquicamente.
- Cada elemento de diseño tiene tanto una interfaz bien definida (para conectarla a otros elementos) como una especificación de comportamiento precisa (para simularla).
- Las especificaciones de comportamiento pueden utilizar ya sea un algoritmo o bien una estructura de hardware real para definir la operación de un elemento. Por ejemplo, un elemento puede ser definido inicialmente por un algoritmo, para permitir la verificación del diseño de elementos de mayor nivel que utiliza; posteriormente, la definición algorítmica puede ser reemplazada por una estructura de hardware.
- La concurrencia, temporización y señales de reloj puede ser todas modeladas. VHDL maneja estructuras de circuitos secuenciales asincrónicos además de sincrónicos.
- La operación lógica y comportamiento de temporización de un diseño pueden simularse.

De este modo, VHDL comenzó como un lenguaje de modelado y documentación, permitiendo que el comportamiento de diseños de sistemas digitales fuera especificado y simulado con precisión.

herramientas de síntesis VHDL

Mientras que el lenguaje VHDL y el entorno de simulación eran importantes innovaciones por sí mismas, la utilidad y popularidad de VHDL dieron un salto cuántico con el desarrollo comercial de *herramientas de síntesis VHDL*. Estos programas pueden crear estructuras de circuito lógico directamente de descripciones de comportamiento VHDL. Si se utiliza VHDL, se puede diseñar, simular y sintetizar cualquier cosa desde un circuito combinacional simple hasta un sistema microprocesador completo en un chip.

VHDL-87
VHDL-93

VHDL fue estandarizado por el IEEE en 1987 (*VHDL-87*) y extendido en 1993 (*VHDL-93*). En esta sección veremos un subconjunto de características del lenguaje que son legales bajo cualquier estándar. Describiremos características adicionales para el diseño lógico secuencial en la sección 7.12.

4.7.1 Flujo de diseño

flujo de diseño

Es útil comprender todo el ambiente de diseño VHDL antes de saltar al lenguaje mismo. Hay varios pasos en un proceso de diseño basado en VHDL, frecuentemente denominado el *flujo de diseño*. Estos pasos son aplicables a cualquier proceso de diseño basado en VHDL y se resumen en la figura 4-50 de la página 266.

EL SIGNIFICADO DE VHDL

"VHDL" viene de "VHSIC Hardware Description Language" (Lenguaje de Descripción de Hardware VHSIC). A su vez, VHSIC quiere decir "Very High Speed Integrated Circuit" (Circuito Integrado de Muy Alta Velocidad), lo que fue un programa del Departamento de Defensa de Estados Unidos para estimular la investigación sobre tecnología de CI de alto rendimiento (¡empleando Cantidades Muy Saludables de Dinero en Efectivo Instantáneo: "Very Healthy Sums of Instant Cash"!).

VERILOG Y VHDL

Aproximadamente al mismo tiempo que VHDL estaba desarrollándose, apareció un lenguaje de diseño de hardware diferente en escena. *Verilog HDL*, o simplemente *Verilog*, fue presentado por Gateway Design Automation en 1984 como un lenguaje de simulación y descripción de hardware propietario. La introducción subsiguiente de herramientas de síntesis basadas en Verilog en 1988 por Sinopsis y la adquisición de Gateway por Cadence Design Systems fue una combinación ganadora que condujo al uso generalizado del lenguaje.

Actualmente, VHDL y Verilog disfrutan ambos de un uso generalizado y comparten el mercado de la síntesis lógica aproximadamente en un 50% cada uno. Verilog tiene sus raíces sintácticas en C y en algunos aspectos es un lenguaje más fácil de aprender y usar, mientras que VHDL es más parecido a Ada (un lenguaje de programación de software patrocinado por el Departamento de Defensa) y tiene más características que respaldan el desarrollo de proyectos grandes.

Al comparar los pros y los contras de un lenguaje contra el otro, los autores David Pellerin y Douglas Taylor lo explican en la forma adecuada en su libro, *VHDL Made Easy!* (Prentice Hall, 1997): Ambos lenguajes son fáciles de aprender y difíciles de dominar. Y una vez que usted haya aprendido alguno, no tendrá problema para hacer la transición al otro.

El denominado "componente frontal" o "interfaz de usuario" comienza con averiguar el enfoque básico y bloques de construcción al nivel del diagrama de bloques. Los diseños lógicos grandes, como programas de software, son por lo regular jerárquicos, y VHDL proporciona un buen marco para definir módulos y sus interfaces y completar los detalles más adelante.

En el paso de *ajuste*, una herramienta de ajuste o *ajustador* mapea las primitivas sintetizadas o componentes con relación a los recursos disponibles del dispositivo. Para un PLD o CPLD, esto puede significar la asignación de ecuaciones a elementos AND-OR que están disponibles. Para un ASIC, puede significar dejar las compuertas individuales en un patrón y hallar formas para conectarlas dentro de las restricciones físicas del ASIC; esto se conoce como el proceso de *ubicación* y *enrutamiento*. El diseñador por lo regular puede especificar restricciones adicionales para esta etapa, tales como la colocación de módulos dentro de un chip o las asignaciones de terminal para las terminales externas de entrada y salida.

editor de texto VHDL

Una vez que se ha escrito algo de código, usted querrá compilarlo, naturalmente. Un *compilador VHDL* analiza su código por si tuviera errores de sintaxis y también verifica que sea compatible con otros módulos de los que depende. También crea la información interna que es necesaria para que un simulador procese su diseño posteriormente. Como en otros esfuerzos de programación, usted probablemente no esperaría hasta el final de la codificación para compilar todo su código. Haciendo un segmento a la vez puede evitarse

EL SIGNIFICADO DE VERILOG

"Verilog" no es un acrónimo, pero tiene algunos interesantes anagramas (en inglés), incluyendo "I, Glover" (¿Danny, el actor?), "G.I. lover" y "I grovel". ¡Oh!, supongo que también podría ser una contracción de "VERIficación de LÓGica".

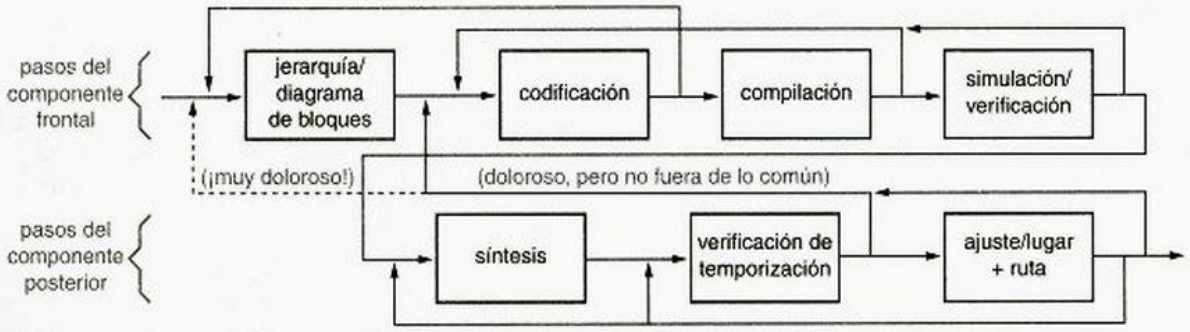


Figura 4-50 Pasos en un flujo de diseño VHDL u otro basado en HDL.

la proliferación de errores de sintaxis, nombres inconsistentes y así sucesivamente, y ciertamente puede darle a usted un sentimiento muy necesario de progreso cuando el final del proyecto ¡parezca inalcanzable!

simulador VHDL

Quizá el paso más satisfactorio viene a continuación: la simulación. Un *simulador VHDL* le permite definir y aplicar entradas a su diseño, y observar sus salidas, sin tener en ningún momento que construir el circuito físico. En proyectos pequeños, el tipo que usted puede hacer como trabajo en casa en una clase de diseño digital, probablemente generaría entradas y observaría salidas manualmente. Pero para proyectos grandes, VHDL le proporciona la capacidad de crear "bancos de prueba" que automáticamente aplican entradas y las comparan con las salidas esperadas.

verificación

En realidad, la simulación es apenas una pieza de un paso mayor conocido como *verificación*. Seguro, es satisfactorio observar que su circuito simulado produce salidas simuladas, pero el propósito de la simulación es mayor: *verificar* que el circuito trabaje como se desea. En un proyecto típicamente grande, una cantidad de esfuerzo sustancial se gasta tanto durante como después de la etapa de codificación para definir casos de prueba que ejerciten el circuito en una amplia gama de condiciones de operaciones lógicas. El hecho de encontrar los errores del diseño en esta etapa es de gran importancia: si los errores se descubren más adelante, todos los llamados pasos del "componente posterior" deben por lo regular ser repetidos.

verificación funcional

verificación de temporización

Advierta que existen al menos dos dimensiones de verificación. En la *verificación funcional*, estudiamos el funcionamiento lógico del circuito sin tomar en cuenta las consideraciones de temporización; los retardos de compuerta y otros parámetros de temporización se consideran como cero. En la *verificación de temporización*, estudiamos la operación del circuito incluyendo retardos estimados, y verificamos que el arranque, retención y otros requerimientos de temporización para los dispositivos secuenciales como son los biestables o flip-flops se satisfagan. Es costumbre efectuar una verificación *funcional* minuciosa antes de comenzar los pasos del componente posterior. Sin embargo, nuestra habilidad para hacer la verificación de *temporización* en esta etapa con frecuencia es limitada, puesto que la temporización puede ser fuertemente dependiente de los resultados de síntesis y ajuste. Podemos hacer una verificación de temporización preliminar para ganar algo de consuelo con el enfoque total del diseño, pero la verificación de temporización detallada debe esperar hasta el final.

Después de la verificación, estamos listos para movernos a la etapa de "componente posterior". La naturaleza de esta etapa y las herramientas para la misma varían un poco, dependiendo de la tecnología que se va a utilizar en el diseño, pero hay tres pasos básicos. El primero es la *síntesis*, que convierte la descripción VHDL en un conjunto de primitivas o

síntesis

componentes que pueden ensamblarse en la tecnología destino. Por ejemplo, con los PLD o CPLD, la herramienta de síntesis puede generar ecuaciones de suma de productos de dos niveles. Con los ASIC, se puede generar una lista de compuertas y una *lista de red* que especifica cómo deberían interconectarse. El diseñador puede "ayudar" a la herramienta de síntesis al especificar ciertas *restricciones* específicas de la tecnología, tal como la cantidad máxima de número de niveles lógicos o la fuerza de los separadores lógicos que se van a utilizar.

lista de red

restricciones

En el paso de *ajuste*, una herramienta de ajuste o *ajustador* mapea las primitivas sintetizadas o componentes con relación a los recursos disponibles del dispositivo. Para un PLD o CPLD, esto puede significar la asignación de ecuaciones a elementos AND-OR que están disponibles. Para un ASIC, puede significar dejar las compuertas individuales en un patrón y hallar formas para conectarlas dentro de las restricciones físicas del ASIC; esto se conoce como el proceso de *ubicación* y *enrutamiento*. El diseñador por lo regular puede especificar restricciones adicionales para esta etapa, tales como la colocación de módulos dentro de un chip o las asignaciones de terminal para las terminales externas de entrada y salida.

ajuste
ajustador

ubicación y enrutamiento

El paso "final" es la verificación de temporización del circuito que está recién ajustado. En esta etapa aparece el retardo de las señales en el circuito real debido a la longitud de los alambres, las cargas eléctricas y otros factores que se pueden calcular con una precisión razonable. Es habitual durante este paso aplicar los mismos casos de prueba que se utilizaron en la verificación funcional, pero las pruebas se realizan en el circuito tal y como se va a construir en realidad.

Como sucede en cualquier otro proceso creativo, ocasionalmente se puede llevar dos pasos para adelante y uno para atrás (¡o peor!). Como se infiere de la figura, durante la codificación se pueden encontrar problemas que nos obliguen a regresar sobre nuestros pasos y volver a pensar en nuestra jerarquía, y casi con certeza se tendrán errores de compilación y simulación que nos forzarán a volver a escribir partes del código.

Los problemas más dolorosos son aquellos que encontramos en el componente posterior del flujo de diseño. Por ejemplo, si el diseño sintetizado no ajusta en un FPGA disponible o no satisface los requerimientos de temporización, se puede tener que regresar tan lejos como para volver a pensar todo el enfoque del diseño. Es algo que vale la pena recordar: las herramientas excelentes todavía no sustituyen un pensamiento cuidadoso al principio de un diseño.

¿¡FUNCIONA!?

Como diseñador lógico y constructor de sistemas con experiencia, siempre pensé que conocía lo que significa que alguien exclame que su circuito "¡funciona!" Significa que usted puede ir al laboratorio, alimentar un prototipo sin ver salir humo, presionar el botón de restablecimiento y emplear el osciloscopio o analizador lógico para observar el prototipo en funcionamiento paso a paso.

Pero, al paso de los años, el significado de "funciona" ha cambiado, al menos para algunas personas. Cuando acepté un nuevo trabajo hace unos cuantos años, estaba muy complacido al escuchar que varios ASIC que eran piezas claves de un importante producto nuevo estaban "funcionando". Pero más tarde (poco tiempo después) averigüé que los ASIC funcionaban solamente en la etapa de simulación, y que el equipo de diseño aún tenía por delante varios arduos meses de síntesis, ajuste, verificación de temporización y repeticiones, antes de que pudieran ordenar cualquier prototipo. "¡Funciona!": sí, seguro. Justo como las tareas de mis niños: "¡Ya está terminada!"

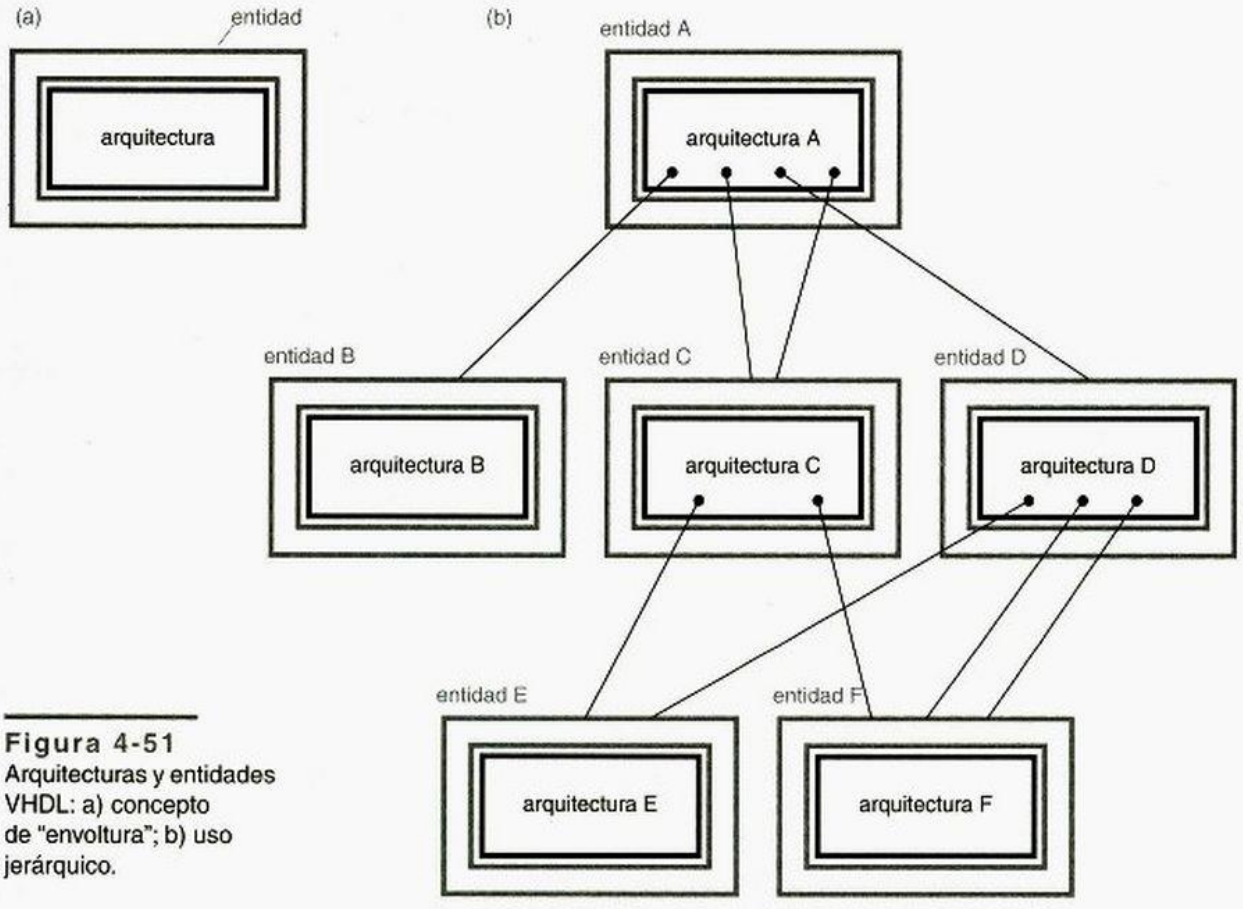


Figura 4-51
Arquitecturas y entidades VHDL: a) concepto de "envoltura"; b) uso jerárquico.

4.7.2 Estructura de programa

entidad
arquitectura

VHDL fue diseñado con principios de programación estructurada en mente, tomando prestadas ideas de los lenguajes de programación de software Pascal y Ada. Una idea clave es definir la interfaz de un módulo de hardware mientras se ocultan sus detalles internos. De este modo, una *entidad* VHDL es simplemente una declaración de las entradas y salidas de un módulo, mientras que una *arquitectura VHDL* es una detallada descripción del comportamiento o estructura interna del módulo.

La figura 4-51(a) ilustra el concepto. Muchos diseñadores gustan de pensar en una declaración de entidad VHDL como una "envoltura" para la arquitectura, ocultando los detalles de lo que se halla dentro mientras proporcionan los "ganchos" o conectores para que otros módulos la usen. Esto forma los fundamentos para el diseño del sistema jerárquico: la arquitectura de una entidad de nivel superior puede utilizar (o "instanciar") otras entidades, mientras oculta los detalles de arquitectura de entidades de nivel más bajo de las de nivel más alto. Como se muestra en (b), una arquitectura de mayor nivel puede hacer uso de una entidad de menor nivel múltiples veces, y múltiples arquitecturas de nivel superior pueden emplear la misma de nivel inferior. En la figura, las arquitecturas B, E y F permanecen aisladas; no utilizan ninguna otra entidad.

¡ADELANTE, CONFIGURE!

En realidad el VHDL nos permite definir múltiples arquitecturas para una sola entidad, y facilita la administración de configuración que permite especificar cuál utilizar durante una compilación particular o ejecución de síntesis. Esto nos deja intentar un enfoque de arquitectura diferente sin desaprovechar u ocultar nuestros otros esfuerzos. Sin embargo, no usaremos o discutiremos más esta facilidad en este texto.

Figura 4-52
Estructura de archivo de programa VHDL.

En el archivo de texto de un programa VHDL, la *declaración de entidad* y la *definición de arquitectura* están separadas, como se ilustra en la figura 4-52. Por ejemplo, la tabla 4-26 es un programa VHDL muy simple para una compuerta "inhibitoria" de 2 entradas. En proyectos grandes, las entidades y arquitecturas se definen en ocasiones en archivos por separado, los cuales el compilador integra de acuerdo con sus nombres declarados.

declaración de entidad
definición de arquitectura

Como otros lenguajes de programación de alto nivel, VHDL generalmente ignora los espacios y los saltos de línea, y éstos pueden proporcionarse como se desee para mejorar la legibilidad. Los *comentarios* comienzan con dos guiones (`--`) y finalizan con el final de la línea.

comentarios

VHDL define muchas cadenas de caracteres especiales, denominadas *palabras reservadas* o *palabras clave*. Nuestro ejemplo incluye varias de ellas: `entity`, `port`, `is`, `in`, `out`, `end`, `architecture`, `begin`, `when`, `else` y `not`. Los *identificadores* definidos por el usuario comienzan con una letra y contienen letras, dígitos y guiones bajos. (Un guión bajo no puede seguir después de otro guión bajo o ser el último carácter en un identificador.) Los identificadores en el ejemplo son Inhibit, `X`, `Y`, `BIT`, `Z` e `Inhibit_arch`. "`BIT`" es un identificador integrado para un tipo predefinido; no se considera una palabra reservada porque puede ser redefinida. Las palabras reservadas y los identificadores no son sensibles a mayúsculas y minúsculas.

palabras reservadas
palabras clave
identificadores

```
entity Inhibit is       -- also know as 'BUT-NOT'
  port (X,Y: in BIT;       -- as in 'X but not Y'
        Z:   out BIT);     -- (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
 Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Tabla 4-26
Programa VHDL para una compuerta "inhibitoria".

Tabla 4-27
Sintaxis de una declaración de entidad VHDL.

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Una declaración de entidad básica tiene la sintaxis mostrada en la tabla 4-27. Aparte de nombrar la entidad, el propósito de la declaración de entidad es definir sus señales de interfaz externa o *puertos* en su parte de *declaración de puertos*. Además de las palabras clave `entity`, `is`, `port` y `end`, una declaración de entidad tiene los elementos siguientes:

puerto
declaración de puerto

entity-name Un identificador seleccionado por el usuario para nombrar la entidad.

signal-name Una lista separada por comas de uno o más identificadores seleccionados por el usuario para nombrar señales de interfaz externa.

mode Una de cuatro palabras reservadas, especificando la dirección de la señal:

`in` La señal es una entrada a la entidad.

`out` La señal es una salida de la entidad. Note que el valor de una señal así no puede ser "leído" dentro de la arquitectura de la entidad, solamente por otras entidades que la utilicen.

`buffer` La señal es una salida de la entidad y su valor también puede ser leído dentro de la arquitectura de la entidad.

`inout` La señal se puede emplear como una entrada o como una salida de la entidad. Este modo es típicamente utilizado para terminales de entrada/salida de tres estados en PLD.

signal-type Un tipo de señal que está integrada o definida por el usuario. Tendremos mucho qué decir acerca de tipos en la siguiente subsección.

Advierta que no hay punto y coma después del *signal-type* final; intercambiando el paréntesis de cierre con el punto y coma después es un error común de sintaxis para los programadores de VHDL principiantes.

definición de arquitectura

Los puertos de una entidad y sus modos y tipos son todo lo que se ve por otros módulos que los utilizan. La operación interna de la entidad se especifica en su *definición de arquitectura*, cuya sintaxis general se muestra en la tabla 4-28. El *entity-name* en esta definición debe ser el mismo que el dado anteriormente en la declaración de entidad. El *architecture-name* es un identificador seleccionado por el usuario, por lo regular relacionado con el nombre de la entidad; puede ser el mismo que el nombre de la entidad si así se desea.

Las señales de interfaz externa (puertos) de una arquitectura se heredan desde la parte de la declaración de puerto de su correspondiente declaración de entidad. Una arquitectura también puede incluir señales y otras declaraciones que sean locales a esa arquitectura, de manera similar a otros lenguajes de alto nivel. Las declaraciones comunes a múltiples entidades pueden hacerse en un "paquete" por separado, utilizado por todas las entidades, como se discute más adelante.

```
architecture architecture-name of entity-name is
  type declarations
  signal declarations
  constant declarations
  function definitions
  procedure definitions
  component declarations
begin
  concurrent-statement
  ...
  concurrent-statement
end architecture-name;
```

Tabla 4-28 Sintaxis de una definición de arquitectura VHDL.

Las declaraciones en la tabla 4-28 pueden aparecer en cualquier orden. En su debido momento discutiremos muchas clases diferentes de declaraciones e instrucciones que pueden aparecer en la definición de arquitectura, pero, para empezar, lo más fácil es la *signal declarations*. Nos da la misma información acerca de una señal que en una declaración de puerto, excepto que no se especifica ningún modo:

declaración de señal

```
signal signal-names : signal-type;
```

Pueden definirse cero o más señales dentro de una arquitectura y más o menos corresponden a los alambres que se identifican en un diagrama lógico. Pueden leerse o escribirse dentro de la definición de la arquitectura y, como otros objetos locales, puede hacerse referencia a ellos solamente dentro de la definición de la arquitectura que los abarca.

Las *variables* VHDL son semejantes a las señales, excepto en que por lo regular no tienen significación física en un circuito. De hecho, note que la tabla 4-28 no tiene provisión para "declaraciones de variables" en una definición de arquitectura. En vez de ello, las variables se utilizan en las funciones, procedimientos y procesos VHDL, cada uno de los cuales discutiremos más adelante. Dentro de estos elementos de programa, la sintaxis de una *declaración de variable* es precisamente como la de una declaración de señal, excepto porque la palabra clave `variable` se emplea:

variable

declaración de variable

```
variable variable-names : variable-type;
```

4.7.3 Tipos y constantes

Todas las señales, variables y constantes en un programa VHDL deben tener un "tipo" asociado. El *tipo* especifica el conjunto o intervalo de valores que el objeto puede tomar, y también hay típicamente un conjunto de operadores (tales como suma, AND, etc.) asociados con un tipo dado.

tipo

VHDL tiene apenas unos cuantos *tipos predefinidos*, enumerados en la tabla 4-29. En el resto de este libro, los únicos tipos predefinidos que utilizaremos son `integer`, `character` y `boolean`. Usted podría pensar que los tipos con nombres "`bit`" y "`bit_vector`" serían esenciales en el diseño digital, pero resulta que las versiones definidas por el usuario de estos tipos son más útiles, como discutiremos en breve.

tipos predefinidos

`bit`	`character`	`severity_level`
`bit_vector`	`integer`	`string`
`boolean`	`real`	`time`

Tabla 4-29 Tipos predefinidos de VHDL.

Tabla 4-30
Operadores predefinidos para los tipos `integer` y `boolean` de VHDL.

Operadores `integer`		***Operadores*** `boolean`	
+	suma	and	AND
-	resta	or	OR
*	multiplicación	nand	NAND
/	división	nor	NOR
mod	división entera	xor	OR Exclusivo
rem	residuo	xnor	NOR Exclusivo
abs	valor absoluto	not	complementación
**	exponenciación		

integer

boolean
true (verdadero), false (falso)
character (carácter)

El tipo *integer (entero)* se define como el intervalo de enteros que incluye por lo menos el intervalo desde –2,147,483,647 hasta +2,147,483,647 ($-2^{31} + 1$ hasta $+2^{31} - 1$); las implementaciones de VHDL pueden extender este intervalo. El tipo *boolean (booleano)* tiene dos valores, *true (verdadero)* y *false (falso)*. El tipo *character (carácter)* contiene todos los caracteres en el conjunto de caracteres ISO de 8 bits; los primeros 128 son los caracteres ASCII. Los operadores integrados para los tipos integer y boolean se enumeran en la tabla 4-30.

tipo definido por el usuario
tipo enumerado

Los tipos más comúnmente utilizados en los programas VHDL típicos son los *tipos definidos por el usuario*, y el más común de estos son los *tipos enumerados* que son definidos al enumerar sus valores. Los tipos predefinidos boolean y character son tipos enumerados. Una declaración de tipo para un tipo enumerado tiene el formato mostrado en la primera línea de la tabla 4-31. Aquí, *value-list* es una lista separada por comas (de enumeración) de todos los valores posibles del tipo. Los valores pueden ser identificadores definidos por el usuario o caracteres (donde un "carácter" es un carácter

Tabla 4-31
Sintaxis de declaraciones de constantes y tipo VHDL.

```
type type-name is (value-list);

subtype subtype-name is type-name start to end;
subtype subtype-name is type-name start downto end;

constant constant-name: type-name := value;
```

FUERTE CONCORDANCIA DE TIPO

A diferencia de C, VHDL es un lenguaje "con fuerte concordancia de tipo". Esto significa que el compilador no le permite a usted asignar un valor a una señal o variable a menos que el tipo del valor coincida precisamente con el tipo declarado de la señal o variable.

La concordancia fuerte de tipo es al mismo tiempo una bendición y una maldición. Hace que sus programas sean más confiables y fáciles de depurar, porque hace difícil que usted cometa "errores tontos" donde se asigna un valor del tipo o tamaño erróneos. Por otro lado, puede ser exasperante a veces. Incluso las operaciones simples, tales como volver a interpretar una señal de 2 bits como un entero (por ejemplo, para seleccionar uno de cuatro resultados en una instrucción "case"), puede requerir que usted llame a una función de conversión de tipo explícitamente.

```
type STD_ULOGIC is ( 'U',  -- No inicializado
                     'X',  -- Forzar desconocido
                     '0',  -- Forzar  0
                     '1',  -- Forzar  1
                     'Z',  -- Alta impedancia
                     'W',  -- Débil Desconocido
                     'L',  -- Débil     0
                     'H',  -- Débil     1
                     '-'   -- Sin importancia
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;
```

Tabla 4-32 Definición del tipo `std_logic` VHDL (véase la sección 5.6.4 para una discusión de "`resolved`").

ISO encerrado en comillas simples). El primer estilo se utiliza con mayor frecuencia para definir casos o estados para una máquina de estado, por ejemplo,

```
type traffic_light_state is (reset, stop, wait, go);
```

El segundo estilo se emplea en el caso muy importante de un tipo lógico estándar definido por el usuario *std_logic*, mostrado en la tabla 4-32 y parte del paquete estándar IEEE 1164, discutido en la sección 4.7.5. Este tipo incluye no solamente '0' y '1', sino otros siete valores que se han encontrado útiles al simular una señal lógica (bit) en un circuito lógico real, como se explica con más detalle en la sección 5.6.4.

std_logic

VHDL también permite a los usuarios crear *subtipos* de un tipo, aplicando la sintaxis que se muestra en la tabla 4-31. Los valores en el subtipo deben ser un intervalo contiguo de valores del tipo base, desde *start* hasta *end*. Para un tipo enumerado, "contiguo" se refiere a las posiciones en la *value-list* definida original. Algunos ejemplos de definiciones de subtipo se muestran a continuación:

subtipos

```
subtype twoval_logic is std_logic range '0' to '1';
subtype fourval_logic is std_logic range 'X' to 'Z';
subtype negint is integer range -2147483647 to -1;
subtype bitnum is integer range 31 downto 0;
```

Note que el orden de un intervalo puede especificarse en forma ascendente o descendente, dependiendo de si se emplea *to* o *downto*. Existen ciertos atributos de subtipos para los cuales esta distinción es significativa, pero no los utilizaremos en este libro y ya no lo estudiaremos.

to
downto

¡QUÉ CARÁCTER! Quizás esté sorprendido y se preguntará por qué los valores en el tipo `std_logic` se definen como caracteres en vez de identificadores de una letra. Ciertamente, "U", "X" y demás serían más fáciles de teclear que "'U'", "'X'", y así sucesivamente. Bien, eso requeriría un identificador de una letra aparte de "-" para usarse para los casos "sin importancia", pero ése no es un problema mayor. La razón principal para utilizar caracteres es que "0" y "1" no podrían ser empleados, debido a que son reconocidos como constantes enteras. Esto nos regresa a la concordancia de tipo de VHDL; no se consideraba aconsejable permitir que al compilador efectuara una conversión de tipo automático dependiendo del contexto.

Tabla 4-33
Sintaxis de las declaraciones de arreglo de VHDL.

```
type type-name is array (start to end) of element-type;
type type-name is array (start downto end) of element-type;
type type-name is array (range-type) of element-type;
type type-name is array (range-type range start to end) of element-type;
type type-name is array (range-type range start downto end) of element-type;
```

VHDL tiene dos subtipos `integer` predefinidos, que se definen a continuación:

```
subtype natural is integer range 0 to highest-integer;
subtype positive is integer range 1 to highest-integer;
```

constantes
declaración de constante

Las *constantes* contribuyen a dar legibilidad, mantenimiento y portabilidad a los programas en cualquier lenguaje. La sintaxis de una *declaración de constante* en VHDL se muestra en la última línea de la tabla 4-31; se muestran ejemplos enseguida:

```
constant BUS_SIZE: integer := 32;     -- ancho del componente
constant MSB: integer := BUS_SIZE-1; -- número de bit del MSB
constant Z: character := 'Z';          -- sinónimo para el valor Hi-Z
```

Note que el valor de una constante puede ser una expresión simple. Las constantes pueden utilizarse en cualquier sitio en que el valor correspondiente pueda ser empleado, y pueden ubicarse para un uso especialmente adecuado en las definiciones de tipo, como lo mostraremos pronto.

tipos arreglo
arreglo
índice de arreglo

Otra categoría muy importante de tipos definidos por el usuario son los *tipos arreglo*. Como otros lenguajes, VHDL define un *arreglo* como un conjunto ordenado de elementos del mismo tipo, donde cada elemento es seleccionado por un *índice de arreglo*. La tabla 4-33 muestra varias versiones de la sintaxis para la declaración de un arreglo en VHDL. En las primeras dos versiones, *start* y *end* son enteros que definen el intervalo posible del índice del arreglo y de aquí el número total de elementos del arreglo. En las últimas tres versiones, la totalidad o un subconjunto de los valores de un tipo existente (*tipo intervalo o tipo rango*) son el intervalo o rango del índice de arreglo.

Los ejemplos de las declaraciones de arreglo se proporcionan en la tabla 4-34. El primer par de ejemplos son muy ordinarios y muestran tanto intervalos ascendentes como descendentes. El ejemplo siguiente muestra cómo una constante, `WORD_LEN`, puede usarse con una declaración de arreglo, mostrando también que un valor de intervalo puede ser una

ACTOS NO NATURALES

Aunque VHDL define el subtipo "natural" estando compuesto de los enteros no negativos y empezando desde el 0, la mayoría de los matemáticos consideran y definen los números naturales comenzando desde el 1. Después de todo, desde los inicios de la historia las personas empezaron contando desde 1, y el concepto del "0" llegó mucho más tarde. Con todo, hay algo de discusión y quizás controversia acerca del tema, especialmente a medida que la edad de la computación ha permitido a más de nosotros pensar en el número 0 como el número de partida. Para lo último en elucubraciones sobre el tema, busque en la Web "natural numbers" ("números naturales").

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

Tabla 4-34
Ejemplos de declaraciones de arreglo VHDL.

expresión simple. El tercer ejemplo muestra que un elemento de arreglo puede ser por sí mismo un arreglo, creando así un arreglo bidimensional. El último ejemplo muestra que un tipo enumerado (o un subtipo) puede especificarse como el intervalo del elemento de arreglo; el arreglo en este ejemplo tiene cuatro elementos, con base en nuestra definición previa de `traffic_light_state`.

Se considera que los elementos de arreglo están ordenados de izquierda a derecha, en la misma dirección del intervalo de índice. De este modo, los elementos que están en la extrema izquierda de los arreglos de tipos `monthly_count`, `byte`, `word`, `reg_file` y `statecount`, tienen índices 1, 7, 31, 1 y `reset`, respectivamente.

Dentro de las instrucciones de programas VHDL, se tiene acceso a los elementos de arreglo individuales utilizando el nombre del arreglo y el índice del elemento entre paréntesis. Por ejemplo, si `M`, `B`, `W`, `R` y `S` son señales de variables de los cinco tipos de arreglo que se definen en la tabla 4-34, entonces todos estos elementos `M(11)`, `B(5)`, `W(WORD_LEN-5)`, `R(1,0)`, `R(1)` y `S(reset)` son válidos.

literal de arreglo

Las *literales de arreglo* pueden especificarse al enumerar los valores de los elementos entre paréntesis. Por ejemplo, la variable byte `B` podría establecerse completamente con unos mediante la instrucción siguiente

```
B := ('1','1','1','1','1','1','1','1');
```

VHDL también tiene una notación abreviada que permite especificar valores mediante el índice. Por ejemplo, para establecer toda la variable `word` `W` a unos excepto para los ceros en el LSB de cada byte, se puede escribir

```
W := (0=>'0',8=>'0',16=>'0',24=>'0',others=>'1');
```

others

Los métodos que se acaban de describir trabajan para arreglos con cualquier *element-type*, pero la manera más fácil de escribir una literal de un tipo de arreglo `STD_LOGIC` es emplear una "cadena de caracteres". Una *cadena* VHDL es una secuencia de caracteres ISO encerrados entre comillas dobles, tal como "`Hola allá`". Una cadena es precisamente un arreglo de caracteres; como resultado, a un arreglo `STD_LOGIC` de una longitud dada se puede asignar el valor de una cadena de la misma longitud, mientras los caracteres en dicha cadena sean tomados del conjunto de nueve caracteres definidos como los posibles valores de los elementos de `STD_LOGIC`: `'0'`, `'1'`, `'U'`, y así sucesivamente. De esta forma, los dos ejemplos anteriores pueden volver a escribirse como sigue:

cadena de caracteres

```
B := "11111111";
W := "11111110111111101111111011111110";
```

porción de arreglo

También es posible hacer referencia a un subconjunto contiguo o *porción* de un arreglo al especificar los índices de inicio y final del subconjunto, por ejemplo, `M(6 to 9)`, `B(3 downto 0)`, `W(15 downto 8)`, `R(2,7 downto 0)`, `R(1 to 2)`, `S(stop to go)`. Note que la dirección de la porción debe ser la misma que la del arreglo original.

operador de concatenación &

Finalmente, usted puede combinar arreglos o elementos de arreglos empleando el *operador de concatenación &*, el cual vincula arreglos y elementos en el orden escrito, de izquierda a derecha. Por ejemplo, `'0'&'1'&"1Z"` es equivalente a `'011Z'`, y la expresión `B(6 downto 0) & B(7)` produce un corrimiento circular a la izquierda de 1 bit del arreglo `B` de 8 bits.

`std_logic_vector`

El tipo de arreglo más importante en programas típicos VHDL es el tipo lógico definido por el usuario estándar 1164 del IEEE `std_logic_vector`, que define un conjunto ordenado de bits `std_logic`. La definición de este tipo es:

```
type STD_LOGIC_VECTOR is array ( natural range <> ) of STD_LOGIC;
```

tipo de arreglo no restringido

Éste es un ejemplo de un *tipo de arreglo no restringido*: el intervalo del arreglo no está especificado, excepto porque debe ser un subintervalo de un tipo definido, en este caso, `natural`. Esta característica de VHDL nos permite desarrollar arquitecturas, funciones y otros elementos de programa en una manera más general, algo independiente del tamaño del arreglo o su intervalo de valores de índice. Un intervalo real se especifica cuando a una señal o variable se asigna este tipo. Veremos ejemplos en la siguiente subsección.

4.7.4 Funciones y procedimientos

función
argumentos
resultado

Como una función en un lenguaje de programación de alto nivel, una *función* VHDL acepta un número de *argumentos* y devuelve un *resultado*. Cada uno de los argumentos y el resultado en una definición de función VHDL o llamada de función tienen un tipo determinado.

definición de función
parámetros formales
parámetros reales

La sintaxis de una *definición de función* se muestra en la tabla 4-35. Después de dar el nombre de la función, numera cero o más *parámetros formales* que se utilizan dentro del cuerpo de la función. Cuando se llama la función, los *parámetros reales* en la llamada de función sustituyen a los parámetros formales. Siguiendo la política de con-

Tabla 4-35
Sintaxis de una definición de función VHDL.

```
function function-name (
      signal-names : signal-type;
      signal-names : signal-type;
      ...
      signal-names : signal-type
) return return-type is
   type declarations
   constant declarations
   variable declarations
   function definitions
   procedure definitions
begin
   sequential-statement
   ...
   sequential-statement
end function-name;
```

cordancia fuerte de tipo VHDL, los parámetros reales deben ser el mismo tipo o un subtipo de los parámetros formales. Cuando la función es llamada desde dentro de una arquitectura, se devuelve un valor del tipo *return-type* en lugar de la llamada función.

Como se muestra en la tabla, una función puede definir sus propios tipos, constantes, variables y procedimientos y funciones anidados. Las palabras clave `begin` y `end` encierran una serie de "instrucciones secuenciales" que se ejecutan cuando la función es llamada. Posteriormente analizaremos las diferentes clases de instrucciones secuenciales y sus sintaxis, pero usted debería ser capaz de comprender los ejemplos que mencionamos en esta sección basándose en su experiencia previa en programación.

La arquitectura de "compuerta inhibitoria" de VHDL de la tabla 4-26 de la página 269 está modificada en la tabla 4-36 para usar una función. Dentro de la definición de función, la palabra clave `return` indica cuándo la función debería regresar con quien la llamó, y está seguida por una expresión con el valor que se devolverá al que la llamó. El tipo resultante de esta expresión debe coincidir con el *return-type* en la declaración de función.

return

El paquete lógico estándar IEEE 1164 define muchas funciones que operan sobre los tipos estándar `std_logic` y `std_logic_vector`. Aparte de especificar un número de tipos definidos por el usuario, el paquete también define las operaciones lógicas básicas sobre estos tipos, tales como `and` y `or`. Esto se aprovecha de la capacidad de VHDL para *sobrecargar* operadores. Esta facilidad permite al usuario especificar una función que sea invocada cuando quiera que se utilice un símbolo de operador integrado (`and`, `or`, +, etc.) con un conjunto coincidente de tipos de operando. Puede haber varias definiciones para un símbolo de operador dado; el compilador automáticamente elige la definición que satisface los tipos de operando en cada uso del operador.

sobrecarga de operador

Por ejemplo, la tabla 4-37 contiene código, tomado del paquete IEEE, que muestra cómo la operación "and" se define para operandos `std_logic`. Este código puede parecer complicado, pero ya hemos presentado todos los elementos básicos del lenguaje que utiliza (excepto por "`resolved`", que describimos en conexión con la lógica de tres estados en la sección 5.6.4).

Las entradas a la función pueden ser del tipo `std_ulogic` o su subtipo `std_logic`. Otro subtipo `UX01` es definido para su uso como el tipo de retorno de la función; incluso si una de las entradas "and" fuera un valor no lógico (`'Z'`, `'W'`, etc.), la función regresará uno de solamente cuatro valores posibles. El tipo `stdlogic_table` define un

Tabla 4-36
Programa VHDL para una función de "inhibición".

```
architecture Inhibit_archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;

begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

Tabla 4-37
Definiciones relacionadas a la operación "and" en valores STD_LOGIC del IEEE 1164.

```
SUBTYPE UX01 IS resolved std_ulogic RANGE 'U' TO '1';
                                                -- ('U','X','0','1')
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

-- truth table for 'and' function
CONSTANT and_table : stdlogic_table := (
--      ---------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -      |   |
--      ---------------------------------------------------
        ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
);

FUNCTION "and" ( L : std_ulogic; R : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (and_table(L, R));
END 'and';
```

arreglo bidimensional de 9×9 indexado por un par de valores `std_ulogic`. Para la `and_table`, las entradas de la tabla se acomodan de modo que si cualquier índice es `'0'` o `'L'` (un `'0'` débil), la entrada es `'0'`. Una entrada `'1'` se encuentra solamente si ambas entradas son `'1'` o `'H'` (un débil `'1'`). De otro modo, aparece una entrada `'U'` o `'X'`.

En la definición de función misma, las dobles comillas alrededor del nombre de la función indican sobrecarga de operador. La parte "ejecutable" de la función es una instrucción simple que devuelve el elemento de tabla que está indexado por ambas entradas, `L` y `R`, de la función "`and`".

Debido a los requerimientos de concordancia fuerte de tipo de VHDL, con frecuencia es necesario convertir una señal de un tipo a otro, y el paquete IEEE 1164 contiene varias funciones de conversión: por ejemplo, de `BIT` a `STD_LOGIC` o viceversa. Una conversión comúnmente necesaria es del `STD_LOGIC_VECTOR` a un valor entero correspondiente. IEEE 1164 no incluye una función de conversión así, porque diseños diferentes pueden necesitar el uso de diferentes interpretaciones de números: por ejemplo, con signo contra sin signo. Sin embargo, podemos escribir nuestra propia función de conversión como se muestra en la tabla 4-38.

La función `CONV_INTEGER` utiliza un algoritmo de iteración simple equivalente a la fórmula de expansión anidada de la página 30. No describiremos las instrucciones `FOR`, `CASE` y `WHEN` que utiliza hasta la sección 4.7.8, pero usted debería captar la idea.

instrucción `null`

La *instrucción* `null` *(nula)* es fácil: significa "no hacer nada". El intervalo del ciclo `FOR` está especificado por "`X'range`", donde la comilla simple o apóstrofo después de un nombre de señal quiere decir "atributo" y *range* es un identificador de atributo integrado

atributo `range`

```
function CONV_INTEGER (X: STD_LOGIC_VECTOR) return INTEGER is
  variable RESULT: INTEGER;
begin
  RESULT := 0;
  for i in X'range loop
    RESULT := RESULT * 2;
    case X(i) is
      when '0' | 'L'  => null;
      when '1' | 'H'  => RESULT := RESULT + 1;
      when others     => null;
    end case;
  end loop;
  return RESULT;
end CONV_INTEGER;
```

Tabla 4-38 Función VHDL para conversión de STD_LOGIC_VECTOR a INTEGER.

que se aplica solamente a arreglos y significa "el intervalo ("rango") de este índice de arreglo, de izquierda a derecha".

En la otra dirección, podemos convertir un entero a un STD_LOGIC_VECTOR como se muestra en la tabla 4-39. Aquí se debe especificar no solamente el valor entero que será convertido (ARG), sino también el número de bits en el resultado deseado (SIZE). Note que la función declara una variable local "result", un STD_LOGIC_VECTOR cuyo intervalo de índice es dependiente del SIZE (tamaño). Por esta razón, SIZE debe ser una constante u otro valor que sea conocido cuando CONV_STD_LOGIC_VECTOR se compile. Para realizar la conversión, la función utiliza el algoritmo de división sucesiva que fue descrito en la página 30.

procedimiento

Un *procedimiento* VHDL es semejante a una función, excepto que no regresa un resultado. Mientras que una llamada de función puede ser empleada en lugar de una expresión, una llamada de procedimiento puede ser utilizada en lugar de una instrucción. Los procedimientos de VHDL permiten que sus argumentos sean especificados con tipo out o inout, de modo que es en realidad posible para un procedimiento "devolver" un resultado. Sin embargo, no empleamos procedimientos VHDL en el resto del libro, así que ya no discutiremos esto.

```
function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
    return STD_LOGIC_VECTOR is
  variable result: STD_LOGIC_VECTOR (SIZE-1 downto 0);
  variable temp: integer;
begin
  temp := ARG;
  for i in 0 to SIZE-1 loop
    if (temp mod 2) = 1 then result(i) := '1';
    else result(i) := '0';
    end if;
    temp := temp / 2;
  end loop;
  return result;
end;
```

Tabla 4-39 Función VHDL para conversión de INTEGER a STD_LOGIC_VECTOR.

4.7.5 Biblioteca y paquetes

biblioteca

Una *biblioteca* VHDL es un lugar donde el compilador VHDL almacena información acerca de un proyecto de diseño particular, incluyendo archivos intermedios que se utilizan en el análisis, simulación y síntesis del diseño. La ubicación de la biblioteca dentro de un sistema de archivos de la computadora anfitriona es dependiente de la implementación. Para un diseño VHDL dado, el compilador automáticamente crea y utiliza una biblioteca llamada "`work`".

Un diseño completo VHDL por lo regular tiene múltiples archivos, cada uno con diferentes unidades de diseño, incluyendo entidades y arquitecturas. Cuando el compilador VHDL analiza cada archivo en el diseño, coloca los resultados en la biblioteca "`work`", y también busca esta biblioteca para definiciones necesarias, tales como otras entidades. Debido a esta característica, un diseño extenso puede ser fragmentado en múltiples archivos, y todavía el compilador encontrará referencias externas cuando las necesite.

No toda la información necesaria en un diseño puede estar en la biblioteca "`work`". Por ejemplo, un diseñador puede depender de definiciones comunes o módulos funcionales a través de una familia de proyectos diferentes. Cada proyecto tiene su propia biblioteca "`work`" (por lo general un subdirectorio dentro del directorio total del proyecto), pero debe también referirse a una biblioteca común que contenga las definiciones compartidas. Incluso los proyectos pequeños pueden utilizar una biblioteca estándar tal como la contenida en las definiciones estándar IEEE. El diseñador puede especificar el nombre de una biblioteca así empleando una cláusula `library` al comienzo del archivo de diseño. Por ejemplo, podemos especificar la biblioteca IEEE:

cláusula `library`

```
library ieee;
```

La cláusula "`library work;`" se incluye de manera implícita al principio de todo archivo de diseño VHDL.

La especificación de un nombre de biblioteca en un diseño nos da acceso a cualquiera de las entidades y arquitecturas previamente analizadas almacenadas en la biblioteca, pero no nos da acceso a las definiciones de tipo y semejantes. Ésta es la función de "paquetes" y "cláusulas `use`", que se describen a continuación.

paquete

Un *paquete* VHDL es un archivo que contiene definiciones de objetos que pueden ser utilizados en otros programas. La clase de objetos que pueden ser puestos en un paquete incluye declaraciones de señal, tipo, constante, función, procedimiento y componentes.

Las señales que se definen en un paquete son señales "globales", disponibles para cualquier entidad VHDL que utilice ese paquete. Los tipos y constantes que se definen en un paquete son conocidos en cualquier archivo que haga uso de ese paquete Del mismo modo, las funciones y procedimientos definidos en un paquete pueden ser llamados en archivos que utilizan el paquete y los componentes (que se describen en la subsección siguiente) pueden ser "instanciados" en arquitecturas que hacen uso del paquete.

cláusula `use`

Un diseño puede "usar" un paquete al incluir una *cláusula* `use` al principio del archivo de diseño. Por ejemplo, para emplear todas las definiciones en el paquete estándar IEEE1164, escribiríamos

```
use ieee.std_logic_1164.all;
```

Aquí, "ieee" es el nombre de la biblioteca que ha sido previamente dada en una cláusula library. Dentro de esta biblioteca, el archivo llamado "std_logic_1164" contiene las definiciones deseadas. El sufijo "all" le dice al compilador que emplee todas las definiciones en este archivo. En vez de "all", se puede escribir el nombre de un objeto particular para usar su definición, por ejemplo,

```
use ieee.std_logic_1164.std_ulogic
```

Esta cláusula haría disponible la definición del tipo std_ulogic en la tabla 4-32 de la página 273, sin todos los tipos y funciones relacionados. Sin embargo, pueden escribirse múltiples cláusulas "use" para emplear definiciones adicionales.

La definición de paquetes no está limitada a cuerpos estándares. Cualquiera puede escribir un paquete, utilizando la sintaxis mostrada en la tabla 4-40. Todos los objetos declarados entre "package" y la primera instrucción "end" son visibles en cualquier archivo de diseño que utilice el paquete; los objetos que siguen a la palabra clave "package body" son locales. En particular, note que la primera parte incluye "declaraciones de funciones", no definiciones. Una *declaración de función* enumera solamente el nombre de la función, argumentos y tipo, hasta pero sin incluir la palabra clave "is" en la tabla 4-35 de la página 276. La definición de función completa está determinada en el cuerpo del paquete y no es visible para los usuarios de la función.

declaración de función

ESTÁNDARES VHDL IEEE

VHDL tiene excelentes capacidades para extender sus tipos de datos y funciones. Esto es importante, porque BIT y BIT_VECTOR integrados en el lenguaje en realidad son bastante poco adecuados para modelar circuitos reales que también manejen señales de tres estados, desconocidas, sin importancia y de intensidad variable.

Como resultado, poco después de que el lenguaje fue formalizado como el estándar IEEE 1076, los representantes comerciales comenzaron a introducir sus propios tipos de datos integrados para tratar con valores lógicos aparte de 0 y 1. Naturalmente, cada representante tenía diferentes definiciones para estos tipos extendidos, creando una potencial "Torre de Babel".

Para evitar esta situación, el IEEE desarrolló al paquete lógico estándar 1164 (std_logic_1164) con un sistema lógico de nueve valores que satisface la mayoría de las necesidades de los diseñadores. Éste fue seguido posteriormente por el estándar 1076-3, discutido en la sección 5.9.6, el cual incluye diversos paquetes con operaciones y tipos estándar para vectores de componentes STD_LOGIC que son interpretados como enteros con o sin signo. Los paquetes incluyen std_logic_arith, std_logic_signed, y std_logic_unsigned.

Al utilizar los estándares IEEE, los diseñadores pueden asegurar un alto grado de transportabilidad y de interoperabilidad entre sus diseños. Esto es de creciente importancia, a medida que la utilización de dispositivos ASIC muy grandes necesita la cooperación, no solamente entre los diseñadores sino también de los vendedores que pueden contribuir cada uno con diferentes piezas de un diseño de "sistema en un chip".

Tabla 4-40
Sintaxis de una definición de paquete VHDL.

```
package package-name is
  type declarations
  signal declarations
  constant declarations
  component declarations
  function declarations
  procedure declarations
end package-name;
package body package-name is
  type declarations
  constant declarations
  function definitions
  procedure definitions
end package-name;
```

4.7.6 Elementos de diseño estructural

Estamos finalmente listos para observar el interior de un diseño VHDL, la porción "ejecutable" de una arquitectura. Recuerde la tabla 4-28 en la página 271 que el cuerpo de una arquitectura es una serie de instrucciones concurrentes. En VHDL, cada *instrucción concurrente* se ejecuta de manera simultánea con otras instrucciones concurrentes en el mismo cuerpo de arquitectura.

instrucción concurrente

Este comportamiento es marcadamente diferente del de las instrucciones en los lenguajes de programación de software convencionales, donde las instrucciones se ejecutan en forma secuencial. Las instrucciones concurrentes son necesarias para simular el comportamiento del hardware, donde los elementos conectados se afectan entre sí de manera continua, no precisamente en segmentos de tiempo particulares, con un orden. De este modo, en un cuerpo de arquitectura VHDL, si la última instrucción actualiza una señal que es utilizada por la primera instrucción, entonces el simulador regresará a esa primera instrucción y actualizará sus resultados de acuerdo con la señal que acaba de cambiar. De hecho, el simulador mantendrá propagándose los cambios y resultados actualizados hasta que el circuito simulado se estabilice; discutiremos esto con más detalle en la sección 4.7.9.

VHDL tiene varias instrucciones concurrentes diferentes, además de un mecanismo para agrupar un conjunto de instrucciones secuenciales para funcionar como una sola instrucción concurrente. Utilizadas en distintas formas, estas instrucciones dan origen a tres estilos algo distintos de descripción y diseño de circuitos, los cuales cubriremos en ésta y las siguientes dos subsecciones.

La más fundamental de las instrucciones concurrentes de VHDL es la instrucción `component`, cuya sintaxis básica se muestra en la tabla 4-41. Aquí, *component-name* es el nombre de una entidad previamente definida que va a emplearse, o "instanciarse" dentro

instanciarse

Tabla 4-41 Sintaxis de una instrucción `component` VHDL.

```
label: component-name port map(signal1, signal2, ..., signaln);

label: component-name port map(port1=>signal1, port2=>signal2, ..., portn=>signaln);
```

```
component component-name
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end component;
```

Tabla 4-42 Sintaxis de una declaración de componente VHDL.

del cuerpo de arquitectura actual. Una instancia de la entidad nombrada es creada para cada instrucción de componente que invoca su nombre, y cada instancia debe nombrarse mediante una *etiqueta* única.

port map

Las palabras clave `port map` introducen una lista que asocia los puertos de la entidad nombrada con señales en la arquitectura actual. La lista puede escribirse ya sea en cualquiera de dos estilos diferentes. El primero es un estilo posicional; como en los lenguajes de programación convencionales, las señales en la lista se encuentran asociadas con los puertos de la entidad en el mismo orden en que aparecen en la definición de la entidad. El segundo es un estilo explícito; cada uno de los puertos de la entidad se conecta a una señal utilizando el operador "=>" y estas asociaciones pueden enumerarse en cualquier orden.

declaración de componente

Antes de ser utilizado en una arquitectura, un componente debe ser declarado en una *declaración de componente* en la definición de la arquitectura (véase la tabla 4-28 de la página 271). Como se muestra en la tabla 4-42, una declaración de componente es, en esencia, lo mismo que la parte de la declaración de puerto de la correspondiente declaración de entidad: numera el nombre, modo y tipo de cada uno de sus puertos.

Los componentes usados en una arquitectura pueden ser los que fueron previamente definidos como parte de un diseño, o pueden ser partes de una librería. La tabla 4-43 es un ejemplo de una entidad VHDL y su arquitectura que emplea componentes, un "detector de números primos" que es estructuralmente idéntico al circuito de nivel de compuerta en la figura 4-30(c) de la página 226. La declaración de entidad nombra las entradas y la salida del circuito. La sección de declaraciones de la arquitectura define todos los nombres de señal y los componentes que son utilizados internamente. Los componentes, `INV`, `AND2`, `AND3`,y `OR4`, están predefinidos en el entorno del diseño en el cual este ejemplo fue creado y compilado (Xilinx Foundation 1.5, consulte las Referencias).

Advierta que las instrucciones de componente en la tabla 4-43 se ejecutan de manera *concurrente*. Incluso si las instrucciones fueran enumeradas en un orden diferente, el mismo circuito sería sintetizado y la operación del circuito simulado sería la misma.

descripción estructural *diseño estructural*

Una arquitectura VHDL que utiliza componentes se denomina a menudo una *descripción estructural* o *diseño estructural*, porque define la estructura de interconexión precisa de señales y entidades que realiza la entidad. En esta consideración, una descripción estructural pura es equivalente a un esquema o lista de red para el circuito.

En algunas aplicaciones es necesario crear múltiples copias de una estructura particular dentro de una arquitectura. Por ejemplo, veremos en la sección 5.10.2 que un "sumador de rizo" de n bits puede ser creado al poner en cascada n "sumadores completos". VHDL incluye una instrucción `generate` que nos permite crear esas estructuras repetitivas utilizando una clase de "ciclo for", sin tener que escribir todas las instancias de componente de manera individual.

Tabla 4-43 Programa estructural VHDL para un detector de números primos.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0);
           P: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

La sintaxis de un ciclo iterativo simple `generate` se muestra en la tabla 4-44. El identificador está implícitamente declarado como una variable con tipo compatible con el *intervalo*. La *instrucción concurrente* se ejecuta una vez para cada posible valor del *identificador* dentro del intervalo, y el *identificador* puede ser empleado dentro de la instrucción concurrente. Por ejemplo, la tabla 4-45 muestra cómo puede ser creado un inversor de 8 bits.

El valor de una constante debe ser conocido en el momento que un programa VHDL es compilado. En muchas aplicaciones es útil diseñar y compilar una entidad y su arquitectura mientras dejamos algunos de sus parámetros, tales como el ancho del bus, sin especificar. La facilidad "genérica" de VHDL nos permite hacer esto.

constante genérica
declaración genérica

Una o más *constantes genéricas* pueden definirse en una declaración de entidad con una *declaración genérica* antes de la declaración de puerto, utilizando la sintaxis mostrada en la tabla 4-46. Cada una de las constantes nombradas pueden utilizarse dentro de la definición de arquitectura para la entidad, y el valor de la constante es diferido hasta que la entidad es instanciada usando una instrucción de componente dentro de otra arquitectura. Dentro de esa instrucción de componente, los valores son asignados a las constantes genéricas utilizando una cláusula *generic map* del mismo estilo que la cláusula `port map`. La tabla 4-47 es un ejemplo que combina las instrucciones `generic` y `generate` para definir un "inversor de bus" con un ancho que es especificado por el usuario. Múltiples copias de este inversor, cada una con un ancho diferente, se instancian en el programa de la tabla 4-48.

generic map

```
label: for identifier in range generate
         concurrent-statement
       end generate;
```

Tabla 4-44
Sintaxis de un ciclo `for-generate` VHDL.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity inv8 is
    port ( X: in STD_LOGIC_VECTOR (1 to 8);
           Y: out STD_LOGIC_VECTOR (1 to 8) );
end inv8;

architecture inv8_arch of inv8 is
component INV por (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
 g1: for b in 1 to 8 generate
       U1: INV port map (X(b), Y(b));
     end generate;
end inv8_arch;
```

Tabla 4-45
Entidad y arquitectura VHDL para un inversor de 8 bits.

```
entity entity-name is
  generic (constant-names : constant-type;
           constant-names : constant-type;
           ...
           constant-names : constant-type);
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Tabla 4-46
Sintaxis de una declaración genérica VHDL dentro de una declaración de entidad.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv is
    generic (WIDTH: positive);
    port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
           Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
end businv;

architecture businv_arch of businv is
component INV por (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
 g1: for b in WIDTH-1 downto 0 generate
       U1: INV port map (X(b), Y(b));
     end generate;
end businv_arch;
```

Tabla 4-47
Entidad y arquitectura VHDL para un inversor de bus de ancho arbitrario.

Tabla 4-48
Entidad y arquitectura VHDL que utiliza el inversor de bus de ancho arbitrario.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv_example is
    port ( IN8: in STD_LOGIC_VECTOR (7 downto 0);
           OUT8: out STD_LOGIC_VECTOR (7 downto 0);
           IN16: in STD_LOGIC_VECTOR (15 downto 0);
           OUT16: out STD_LOGIC_VECTOR (15 downto 0);
           IN32: in STD_LOGIC_VECTOR (31 downto 0);
           OUT32: out STD_LOGIC_VECTOR (31 downto 0) );
end businv_example;

architecture businv_ex_arch of businv_example is
component businv
    generic (WIDTH: positive);
    port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
           Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
end component;
begin
U1: businv generic map (WIDTH=>8) port map (IN8, OUT8);
U2: businv generic map (WIDTH=>16) port map (IN16, OUT16);
U3: businv generic map (WIDTH=>32) port map (IN32, OUT32);
end businv_ex_arch;
```

4.7.7 Elementos de diseño de flujo de datos

Si las instrucciones de componentes fueran sus únicas instrucciones concurrentes, entonces VHDL sería un poco más que un lenguaje de descripción de lista de red jerárquica. Varias instrucciones concurrentes adicionales permiten a VHDL describir un circuito en términos del flujo de datos y operaciones activadas dentro del circuito. Este estilo se denomina una *descripción de flujo de datos* o *diseño de flujo de datos*.

descripción de flujo de datos
diseño de flujo de datos
instrucción de asignación de señal concurrente

Se muestran en la tabla 4-49 dos instrucciones concurrentes adicionales utilizadas en diseños de flujo de datos. La primera de éstas es la que se usa con más frecuencia y se conoce como una *instrucción de asignación de señal concurrente*. Se puede leer esto como "*signal-name* obtiene *expression*". Debido a la concordancia de tipo VHDL, el tipo de *expresión* debe ser compatible con el de *signal-name*. En general, esto significa que cualquiera de los tipos debe ser idéntico o el tipo de *expression* es un subtipo del de *signal-name*. En el caso de los arreglos tanto el tipo de elemento como la longitud deben coincidir; sin embargo, el intervalo del índice y la dirección no necesitan hacerlo.

Tabla 4-49
Sintaxis de las instrucciones de asignación de señal concurrentes.

```
signal-name <= expression;

signal-name <= expression when boolean-expression else
               expression when boolean-expression else
               ...
               expression when boolean-expression else
               expression;
```

```
architecture prime2_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= not N(3)                              and N(0);
  N3L_N2L_N1 <= not N(3) and not N(2) and     N(1)          ;
  N2L_N1_N0  <=              not N(2) and     N(1) and N(0);
  N2_N1L_N0  <=                  N(2) and not N(1) and N(0);
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime2_arch;
```

Tabla 4-50
Arquitectura VHDL de flujo de datos para el detector de números primos.

La tabla 4-50 muestra una arquitectura para la entidad del detector de números primos (tabla 4-43 de la página 284) escrita en el estilo de flujo de datos. En este estilo no mostramos las compuertas explícitas y sus conexiones; en vez de ello, utilizamos los operadores integrados de VHDL `and`, `or` y `not`. (En realidad, estos operadores no se encuentran integrados para señales del tipo `STD_LOGIC`, pero son definidos y sobrecargados por el paquete IEEE 1164.) Note que el operador `not` tiene la precedencia más alta, de modo que no se requiere de paréntesis en torno de subexpresiones como "`not N(3)`" para obtener el resultado deseado.

instrucción de asignación de señal condicional

`when`

`else`

También podemos emplear la segunda, forma *condicional* de la instrucción de asignación de señal concurrente, al aplicar las palabras clave `when` y `else` como se muestra en la tabla 4-49. Aquí, una *expresión booleana* combina términos booleanos individuales utilizando operadores booleanos de VHDL tales como `and`, `or` y `not`. Los términos booleanos son por lo regular variables booleanas o resultados de comparaciones empleando los *operadores relacionales* =, /= (no igual), >, >=, < y <=.

operadores relacionales

=, /=, >, >=, <, <=

La tabla 4-51 es un ejemplo que utiliza instrucciones de asignación concurrente condicional. Cada una de las comparaciones de un bit `STD_LOGIC` individual, tal como `N(3)` se hace contra una literal de carácter `'1'` o `'0'` y devuelve un valor de tipo `boolean`. Estos resultados de la comparación son combinados en la expresión booleana entre las palabras clave `when` y `else` en cada instrucción. Las cláusulas `else` son generalmente requeridas; el conjunto combinado de condiciones en una instrucción simple debería cubrir todas las posibles combinaciones de entrada.

instrucción de asignación de señal seleccionada

Otra clase de instrucción asignación concurrente es la *asignación de señal seleccionada*, cuya sintaxis se muestra en la tabla 4-52. Esta instrucción evalúa la *expression* dada, y cuando el valor satisface una de las *options*, asigna el correspondiente *signal-value* señal al *signal-name*. Las *opciones* en cada cláusula `when` puede ser un valor simple de *expresión* o una lista de valores separados por barras verticales (|). Las *opciones* para

```
architecture prime3_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= '1' when N(3)='0' and N(0)='1' else '0';
  N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
  N2L_N1_N0  <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
  N2_N1L_N0  <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

Tabla 4-51
Arquitectura del detector de números primos utilizando asignaciones condicionales.

Tabla 4-52
Sintaxis de instrucción de asignación de señal seleccionada VHDL.

```
with expression select
  signal-name <= signal-value when choices,
                 signal-value when choices,
                 ...
                 signal-value when choices;
```

toda la instrucción deben ser mutuamente exclusivas y todas inclusivas. La palabra clave `others` puede ser utilizada en la última cláusula `when` para indicar todos los valores de *expresión* que no hayan sido cubiertos.

others

La tabla 4-53 es una arquitectura para el detector de números primos que hace uso de una instrucción de asignación de señal seleccionada. Todas las *opciones* para las cuales `F` es `'1'` podrían haberse escrito en una sola cláusula `when`, pero se muestran múltiples cláusulas sólo por motivos didácticos. En este ejemplo, la instrucción de asignación de señal seleccionada lee algo como una lista del conjunto de iniciación de la función `F`.

Podemos modificar ligeramente la anterior arquitectura para tomar ventaja de la interpretación numérica de `N` en la definición de función. Empleando la función `CONV_INTEGER` que definimos con anterioridad, la tabla 4-54 escribe las *opciones* en términos de enteros, lo que podemos fácilmente ver como primos a medida que se requiera. Podemos pensar de esta versión de la arquitectura como una descripción de "comportamiento", porque describe la función deseada de tal manera que su comportamiento es bastante evidente.

Tabla 4-53
Arquitectura del detector de números primos que utiliza la asignación de señal seleccionada.

```
architecture prime4_arch of prime is
begin
  with N select
    F <= '1' when "0001",
         '1' when "0010",
         '1' when "0011" | "0101" | "0111",
         '1' when "1011" | "1101",
         '0' when others;
end prime4_arch;
```

CUBRIENDO TODOS LOS CASOS

Las asignaciones de señal seleccionadas y condicionales requieren que todas las posibles condiciones sean cubiertas. En una asignación de señal condicional, el "`else` *expression*" final cubre las condiciones perdidas. En una asignación de señal seleccionada, "`others`" puede utilizarse en la cláusula `when` final para captar las condiciones restantes.

En la tabla 4-53, se puede pensar que en vez de escribir "`others`" en la cláusula `when` final, podríamos haber escrito las nueve combinaciones de 4 bits restantes, "`0000`", "`0100`", y así sucesivamente. ¡Pero esto no es cierto! Recuerde que `STD_LOGIC` es un sistema de nueve valores, de modo que un `STD_LOGIC_VECTOR` de 4 bits en realidad tiene 9^4 posibles valores. ¡De modo que "`others`" en este ejemplo se encuentra en realidad cubriendo 6,554 casos!

```
architecture prime5_arch of prime is
begin
  with CONV_INTEGER(N) select
    F <= '1' when 1 | 2 | 3 | 5 | 7 | 11 | 13,
         '0' when others;
end prime5_arch;
```

Tabla 4-54
Una descripción más enfocada al comportamiento del detector de números primos.

4.7.8 Elementos de diseño basado en el comportamiento

Como vimos en el último ejemplo, es posible en ocasiones describir de manera directa el comportamiento de un circuito lógico deseado empleando una instrucción concurrente. Esto es algo bueno, como la capacidad de crear un *diseño de comportamiento* o *descripción de comportamiento* es uno de los beneficios clave de los lenguajes de descripción de hardware en general y de VHDL en particular. Sin embargo, para la mayoría de las descripciones de comportamiento, necesitamos emplear algunos elementos de lenguaje adicionales descritos en esta subsección.

diseño basado en el comportamiento

descripción de comportamiento

El elemento de comportamiento clave de VHDL es el "proceso". Un *proceso* es una colección de instrucciones "secuenciales" (descritas en breve) que se ejecuten en paralelo con otras instrucciones concurrentes y otros procesos. Utilizando un proceso se puede especificar una compleja interacción de señales y eventos en una manera que ejecuta, en un tipo de simulación esencialmente cero durante ésta y da origen a un circuito secuencial o combinacional sintetizado, que efectúa la operación modelada directamente.

proceso

Una *instrucción de proceso* (process) puede utilizarse en cualquier parte en que pueda ser empleda una instrucción concurrente. Se introduce mediante la palabra clave `process` y tiene la sintaxis mostrada en la tabla 4-55. Puesto que una instrucción de proceso está escrita dentro del ámbito de una arquitectura cerrada, tiene visibilidad de los tipos, señales, constantes, funciones y procedimientos que están declarados o son de otra manera visibles en la arquitectura cerrada. Sin embargo, se pueden definir también tipos variables, constantes, funciones y procedimientos que sean locales al proceso.

instrucción de proceso

process

Note que un proceso puede no declarar señales, solamente "variables". Una *variable* VHDL mantiene su estado dentro de un proceso y no es visible fuera de él. Dependiendo de su uso, puede o no dar origen a una señal correspondiente en una realización física del circuito modelado. La sintaxis para definir una variable dentro de un proceso es

```
process (signal-name, signal-name, ..., signal-name)
  type declarations
  variable declarations
  constant declarations
  function definitions
  procedure definitions
begin
  sequential-statement
  ...
  sequential-statement
end process;
```

Tabla 4-55
Sintaxis de una instrucción de proceso VHDL.

similar a la de una declaración de señal dentro de una arquitectura, a excepción de que se utiliza la palabra clave `variable`:

variable

```
variable variable-names : variable-type;
```

proceso de ejecución
proceso suspendido
lista de sensibilidad

Un proceso VHDL siempre se encuentra *en ejecución* o *suspendido*. La lista de señales en la definición del proceso, llamada la *lista de sensibilidad*, determina cuándo el proceso se encuentra en ejecución. Un proceso inicialmente está suspendido: cuando cualquier señal en su lista de sensibilidad cambia de valor, el proceso reanuda la ejecución, comenzando con su primera instrucción secuencial y continuando hasta el final. Si cualquier señal en la lista de sensibilidad cambia de valor como resultado de la ejecución del proceso, se ejecuta de nuevo. Esto continúa hasta que el proceso se ejecuta sin ninguna de estas señales que cambian de valor. En la simulación, todo esto pasa en un tiempo simulado de cero.

Al reiniciarse, un proceso escrito apropiadamente se suspenderá después de una o unas cuantas ejecuciones. Sin embargo, es posible escribir un proceso incorrecto que nunca se suspenda. Por ejemplo, considere un proceso con sólo una instrucción secuencial, "`X <= not X`" y una lista de sensibilidad de "`(X)`". Puesto que `X` cambia a cada paso, el proceso se ejecutará por siempre en tiempo de simulación cero: ¡algo no muy útil! En la práctica, los simuladores tienen sistemas de seguridad que normalmente pueden detectar un comportamiento así de indeseable, terminando el proceso mal comportado después de unas miles de pasadas.

La lista de sensibilidad es opcional; un proceso sin una lista de sensibilidad comienza ejecutándose en el tiempo cero en simulación. Una aplicación de un proceso de esta clase es para generar formas de onda de entrada en un banco de pruebas, como en la tabla 4-65 de la página 296.

instrucción de asignación de señal secuencial

instrucción de asignación de variable

:=

VHDL tiene varias clases de instrucciones secuenciales. La primera es una *instrucción de asignación de señal secuencial*; ésta tiene la misma sintaxis que la versión concurrente (*nombre-señal <= expresión;*), pero esto ocurre dentro del cuerpo de un proceso más que de una arquitectura. Una instrucción análoga para variables es la *instrucción de asignación de variable*, la cual tiene la sintaxis "*variable-name := expression;*". Note que se utiliza un operador de asignación diferente, `:=`, para variables.

Para propósitos didácticos, la arquitectura del flujo de datos del detector de números primos en la tabla 4-50 se vuelve a escribir como un proceso en la tabla 4-56. Note que todavía seguimos trabajando con la misma declaración de entidad original de

Tabla 4-56
Arquitectura VHDL de flujo de datos basada en proceso para el detector de números primos.

```
architecture prime6_arch of prime is
begin
  process(N)
    variable N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
  begin
    N3L_N0     := not N(3)                              and N(0);
    N3L_N2L_N1 := not N(3) and not N(2) and     N(1)          ;
    N2L_N1_N0  :=              not N(2) and     N(1) and N(0);
    N2_N1L_N0  :=                  N(2) and not N(1) and N(0);
    P <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
  end process;
end prime6_arch;
```

COMPORTAMIENTO EXTRAÑO

Recuerde que las instrucciones dentro de un proceso se ejecutan *secuencialmente*. Supongamos que por alguna razón escribimos la última instrucción en la tabla 4-56 (la asignación de señal para F) como la primera. Entonces veríamos un comportamiento bastante extraño de este proceso.

La primera vez que el proceso estuviera ejecutándose, el simulador protestaría que los valores de las variables fueran leídos antes de que algún valor se les asignara. En posteriores reanudaciones, se asignaría un valor a F basado en los valores *previos* de las variables, que se recuerdan mientras que el proceso está suspendido. Los nuevos valores entonces serían asignados a las variables y recordados hasta la siguiente reanudación. De modo que el valor de salida del circuito siempre estaría respondiendo atrasado por un cambio de la entrada.

prime que apareció en la tabla 4-43. En la nueva arquitectura (prime6_arch), tenemos precisamente una instrucción concurrente, la cual es un proceso. La lista de sensibilidad del proceso contiene solamente N, las entradas primarias de la función lógica combinacional deseada. Las salidas de compuerta AND deben ser definidas como variables más que como señales, puesto que las definiciones de señal no se permiten dentro de un proceso. De otro modo, el cuerpo del proceso es muy semejante al de la arquitectura original. De hecho, una herramienta de síntesis típica probablemente crearía el mismo circuito de cualquier descripción.

Otras instrucciones secuenciales, más allá de la asignación simple, nos pueden dar más control creativo para expresar el comportamiento del circuito. La *instrucción* `if`, con la sintaxis mostrada en la tabla 4-57, es probablemente la más familiar de ellas. En la primera y más simple forma de la instrucción, se prueba una *boolean-expression*, y se ejecuta una *sequential-statement* si el valor de la expresión es `true` (verdadero). En la

instrucción if

Tabla 4-57
Sintaxis de una instrucción `if` de VHDL.

```
if boolean-expression then sequential-statement
end if;

if boolean-expression then sequential-statement
else sequential-statement
end if;

if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
...
elsif boolean-expression then sequential-statement
end if;

if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
...
elsif boolean-expression then sequential-statement
else sequential-statement
end if;
```

Tabla 4-58
Arquitectura del detector de números primos que utiliza una instrucción `if`.

```
architecture prime7_arch of prime is
begin
  process(N)
    variable NI: INTEGER;
  begin
    NI := CONV_INTEGER(N);
    if NI=1 or NI=2 then F <= '1';
    elsif NI=3 or NI=5 or NI=7 or NI=11 or NI=13 then F <= '1';
    else F <= '0';
    end if;
  end process;
end prime7_arch;
```

else

segunda forma, hemos agregado una cláusula "*else*" con otra *sequential-statement* que se ejecuta si el valor de la expresión es `false` (falso).

elsif

Para crear instrucciones anidadas `if-then-else`, VHDL utiliza una palabra clave especial *elsif*, que introduce las cláusulas "en el medio". Una *sequential-statement* de la cláusula `elsif` se ejecuta si su *boolean-expression* es `true` y todas las *boolean-expressions* precedentes fueran `false`. La *sequential-statement* de la cláusula `else` final opcional se ejecuta si todas las anteriores *boolean-expressions* fueran `false`.

La tabla 4-58 es una versión de la arquitectura del detector de números primos que emplea una instrucción `if`. Se utiliza una variable local `NI` para mantener una versión entera, convertida, de la entrada `N`, de modo que las comparaciones en la instrucción `if` puedan ser escritas utilizando valores enteros.

Las expresiones booleanas en la tabla 4-58 son del tipo que no se traslapan; es decir, solamente una de ellas es `true` a un tiempo. Para esta aplicación realmente no necesitamos la potencia total de las instrucciones `if` anidadas. De hecho, una máquina de síntesis puede crear un circuito que evalúe las expresiones booleanas en serie, con operación más lenta, lo que podría ser posible de otro modo. Cuando necesitamos seleccionar entre múltiples alternativas basadas en el valor de sólo una señal o expresión, por lo regular una *case statement* es más legible y puede producir un circuito mejor sintetizado.

instrucción case

La tabla 4-59 muestra la sintaxis de una instrucción `case`. Esta instrucción evalúa la *expression* dada, encuentra un valor coincidente en una de las *choices* y ejecuta las correspondientes *sequential-statements*. Advierta que una o más instrucciones secuenciales pueden escribirse para cada conjunto de *choices*. Las *choices* pueden tomar la forma de un valor simple o de múltiples valores separados por barras verticales (`|`). Las *choices* deben ser mutuamente excluyentes e incluir todos los valores posibles del tipo de *expression*; la palabra clave `others` puede emplearse como la última de las *choices* para indicar todos los valores que aún no han sido cubiertos.

Tabla 4-59
Sintaxis de una instrucción `case` de VHDL.

```
case expression is
  when choices => sequential-statements
  ...
  when choices => sequential-statements
end case;
```

```
architecture prime8_arch of prime is
begin
  process(N)
  begin
    case CONV_INTEGER(N) is
      when 1 => F <= '1';
      when 2 => F <= '1';
      when 3 | 5 | 7 | 11 | 13 => F <= '1';
      when others => F <= '0';
    end case;
  end process;
end prime8_arch;
```

Tabla 4-60
Arquitectura del detector de números primos que utiliza una instrucción `case`.

La tabla 4-60 es otra arquitectura más para el detector de números primos, codificada esta vez como una instrucción `case`. Como la versión concurrente, la instrucción `select` en la tabla 4-54 de la página 289, la instrucción `case` hace muy fácil de ver el comportamiento funcional deseado.

instrucción `loop`

Otra importante clase de instrucciones secuenciales la constituyen las *instrucciones* `loop`. La más simple de éstas tiene la sintaxis de la tabla 4-61 y crea un ciclo ("loop") infinito. Aunque los ciclos infinitos son indeseables en los lenguajes de programación de software convencionales, demostraremos en la sección 7.12 cómo un ciclo de tal naturaleza puede ser muy útil en el modelado de hardware.

```
loop
  sequential-statement
  ...
  sequential-statement
end loop;
```

Tabla 4-61
Sintaxis de una instrucción `loop` VHDL básica.

ciclo `for`

Un ciclo más familiar, uno que hemos visto con anterioridad, es el ciclo `for`, con la sintaxis mostrada en la tabla 4-62. Advierta que la variable del ciclo, *identifier*, se declara implícitamente por su aparición en el ciclo `for` y tiene el mismo tipo de *range*. Esta variable puede ser empleada dentro de las instrucciones secuenciales del ciclo y pasa a través de todos los valores de *range*, de izquierda a derecha, una por iteración.

Dos útiles instrucciones secuenciales más que pueden ser ejecutadas dentro de un ciclo son "`exit`" y "`next`". Cuando se ejecuta, `exit` transfiere el control a la instrucción que sigue inmediatamente al final del ciclo. Por el otro lado, `next` provoca que cualquier instrucción restante en el ciclo sea omitida y comience la siguiente iteración del ciclo.

instrucción `exit`
instrucción `next`

```
for identifier in range loop
  sequential-statement
  ...
  sequential-statement
end loop;
```

Tabla 4-62
Sintaxis de un ciclo `for` VHDL.

Tabla 4-63
Arquitectura del detector de números primos que utiliza una instrucción for.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime9 is
    port ( N: in STD_LOGIC_VECTOR (15 downto 0);
           F: out STD_LOGIC );
end prime9;

architecture prime9_arch of prime9 is
begin
  process(N)
  variable NI: INTEGER;
  variable prime: boolean;
  begin
    NI := CONV_INTEGER(N);
    prime := true;
    if NI=1 or NI=2 then null; -- considerar los casos límite (límites)
    else for i in 2 to 253 loop
           if (NI mod i = 0) and (NI /= i) then
             prime := false; exit;
           end if;
         end loop;
    end if;
    if prime then F <= '1'; else F <= '0'; end if;
  end process;
end prime9_arch;
```

Nuestro viejo y conocido detector de números primos está codificado una vez más en la tabla 4-63, para esto aplicamos un ciclo `for`. El punto atractivo acerca de este ejemplo es que es verdaderamente una descripción de comportamiento: en realidad hemos utilizado VHDL para calcular si la entrada `N` es un número primo. También hemos incrementado el tamaño de `N` a 16 bits, precisamente para enfatizar el hecho de que fuéramos capaces de crear un modelo compacto para el circuito sin tener que enumerar de manera explícita cientos de primos.

DISEÑO ERRÓNEO DE UN CIRCUITO

La tabla 4-63 tiene un buen ejemplo de un ciclo `for`, pero es un mal ejemplo de cómo diseñar un circuito. Aunque VHDL es un poderoso lenguaje de programación, las descripciones de diseño que utilizan todo su poder pueden ser poco eficientes o no sintetizables.

El culpable en la tabla 4-63 es el operador `mod`. Esta operación requiere una división entera, y la mayoría de las herramientas de VHDL son incapaces de sintetizar circuitos de división excepto por casos especiales, tales como la división entre una potencia de dos (efectuada como un desplazamiento).

Incluso si las herramientas pudieran sintetizar un divisor, no quisiéramos especificar un detector de números primos de esta manera. ¡La descripción en la tabla 4-63 implica un circuito combinacional, y las herramientas tendrían que crear 252 divisores combinacionales, uno para cada valor de `i`, para "desenrollar" el ciclo `loop` y realizar el circuito!

```
while boolean-expression loop
  sequential-statement
  ...
  sequential-statement
end loop;
```

Tabla 4-64
Sintaxis de un ciclo `while` VHDL.

ciclo `while`

La última clase de instrucción `loop` es el ciclo *`while`*, con la sintaxis que se muestra en la tabla 4-64. En esta forma, *boolean-expression* se prueba antes de cada iteración del ciclo, y el ciclo se ejecuta solamente si el valor de la expresión es `true`.

Podemos utilizar procesos para escribir descripciones de comportamiento tanto de circuitos combinacionales como secuenciales. Muchos ejemplos más de descripciones de circuito combinacional aparecen en las subsecciones VHDL del capítulo 5. Unas cuantas características adicionales de lenguaje son necesarias para describir circuitos secuenciales; de éstos trata la sección 7.12, mientras que los ejemplos secuenciales aparecen en las subsecciones VHDL del capítulo 8.

4.7.9 La dimensión tiempo y simulación

Ninguno de los ejemplos con los que hemos tratado hasta ahora modelan la dimensión de tiempo del funcionamiento del circuito: todo ocurre en un tiempo simulado de cero. Sin embargo, VHDL tiene excelentes facilidades para modelar el tiempo, y es en efecto otra dimensión significativa del lenguaje. En este libro no entraremos en detalles acerca de este tema, pero presentaremos sólo algunas ideas aquí.

palabra clave `after`

VHDL nos permite especificar un retardo de tiempo utilizando la palabra clave *`after`* en cualquier instrucción de asignación de señal, incluyendo asignaciones secuenciales, concurrentes, condicionales y seleccionadas. Por ejemplo, en la arquitectura de compuerta inhibitoria de la tabla 4-26 de la página 269 se podría escribir

```
Z <= '1' after 4 ns when X='1' and Y='0' else '0' after 3 ns;
```

Esto nos permite modelar una compuerta de inhibición que tiene 4 ns de retardo en una transición de salida de 0 a 1 y solamente 3 ns en una transición de 1 a 0. En entornos de diseño ASIC típicos, los modelos VHDL para todos los componentes de bajo nivel en la biblioteca de componentes incluyen tales parámetros de retardo. Empleando estas estimaciones, un simulador VHDL puede predecir el comportamiento de temporización aproximado de un circuito más grande que utilice estos componentes.

instrucción `wait`

Otra manera de invocar la dimensión de tiempo es con *`wait`*, una instrucción secuencial. Ésta puede ser utilizada para suspender un proceso por un periodo especificado de tiempo. La tabla 4-65 es un programa de ejemplo que utiliza la instrucción `wait` para crear formas de onda simuladas, que verifican el funcionamiento de la compuerta de inhibición para cuatro diferentes combinaciones de entrada en pasos de tiempo de 10 ns.

Una vez que tengamos un programa VHDL cuyas sintaxis y semántica sean correctas, se puede utilizar un simulador VHDL para observar su funcionamiento. Aunque no entraremos en detalles, es útil tener un entendimiento básico de cómo trabaja un simulador de esta clase.

Tabla 4-65
Uso de la instrucción `wait` de VHDL para generar formas de onda en un programa de banco de pruebas.

```
entity InhibitTestBench is
end InhibitTestBench;

architecture InhibitTB_arch of InhibitTestBench is
component Inhibit port (X,Y: in BIT; Z: out BIT); end component;
signal XT, YT, ZT: BIT;
begin
  U1: Inhibit port map (XT, YT, ZT);
  process
  begin
    XT <= '0'; YT <= '0';
    wait for 10 ns;
    XT <= '0'; YT <= '1';
    wait for 10 ns;
    XT <= '1'; YT <= '0';
    wait for 10 ns;
    XT <= '1'; YT <= '1';
    wait; -- this suspends the process indefinitely
  end process;
end InhibitTB_arch;
```

tiempo de simulación

El funcionamiento del simulador comienza en un *tiempo de simulación* de cero. En este momento, el simulador inicializa todas las señales a un valor predeterminado (¡del cual usted no debería depender!). También inicializa cualquier señal o variable para las que los valores iniciales hayan sido declarados de manera explícita (no hemos mostrado cómo hacer esto). A continuación, el simulador comienza la ejecución de todos los procesos e instrucciones concurrentes en el diseño.

Naturalmente, el simulador no puede hacer en realidad la simulación de todos los procesos e instrucciones concurrentes simultáneamente, pero puede pretender que lo hace, utilizando una "lista de eventos" basada en el tiempo y una "matriz de sensibilidad de señal". Advierta que cada instrucción concurrente es equivalente a un proceso.

Para simulación de tiempo cero, todos los procesos son programados para la ejecución, y uno de éstos se selecciona. Todas estas instrucciones secuenciales se ejecutan, incluyendo cualquier comportamiento repetitivo que se especifique. Cuando la ejecución de este proceso se completa, otro es seleccionado, y así sucesivamente, hasta que todos los procesos han sido ejecutados. Esto completa un *lazo de simulación*.

lazo de simulación

lista de eventos

Durante su ejecución, un proceso puede asignar nuevos valores a las señales. Los nuevos valores no se asignan inmediatamente; en vez de ello, se colocan en una *lista de eventos* y se programan para llegar a ser efectivos en un cierto tiempo. Si la asignación tiene un tiempo de simulación específico asociado con ella (por ejemplo, después de un retardo especificado por una cláusula `after`), entonces es programado en la lista de eventos para presentarse en ese momento. De otro modo, se supone que ocurren "de inmediato"; sin embargo, en realidad es programado para presentarse en el tiempo de simulación actual más un "retardo delta". El *retardo delta* es un tiempo infinitesimalmente corto, de tal modo que el tiempo de simulación actual más cualquier número de retardos delta todavía es igual al tiempo de simulación actual. Este concepto permite a los procesos ejecutarse múltiples veces si es necesario en tiempo simulado de cero.

retardo delta

Después de que se completa un lazo de simulación, la lista de eventos se explora para la señal o señales que cambien en el siguiente tiempo más próximo en la lista. Esto puede ser tan diminuto como un retardo delta posterior, o puede ser un retardo de circuito real, en cuyo caso el tiempo de simulación se avanza a este tiempo. En cualquier caso, se efectúan los cambios de la señal programada. Algunos procesos son sensibles a las señales cambiantes, como es indicado por la matriz de sensibilidad de señal. Indica que, para cada señal, cuáles procesos tienen esa señal en su lista de sensibilidad. (El proceso equivalente de una instrucción concurrente tiene *todas* estas señales de datos y control en su lista de sensibilidad). Todos los procesos que son sensibles a una señal que acaba de cambiar son programados para su ejecución en el siguiente lazo de simulación, que comienza ahora.

La operación en dos fases del simulador consistente en un lazo de simulación seguido por la exploración de la lista de eventos y la realización de las siguientes asignaciones de señal programadas sigue indefinidamente, hasta que la lista se encuentre vacía. Hasta este punto la simulación está completa.

El mecanismo de la lista de eventos hace posible simular la operación de los procesos concurrentes aun cuando el simulador corra en una sola computadora con un solo hilo de ejecución. Y el mecanismo de retardo delta asegura de correcta operación aun cuando un proceso o conjunto de procesos puedan requerir múltiples ejecuciones, dando vueltas en varios retardos delta, antes de que las señales cambiantes se adapten a un valor estable. Este mecanismo también se utiliza para detectar procesos incontrolados (tales como "`X <= not X`"); si se presenta un millar de lazos de simulación sobre un millar de retardos delta sin avanzar el tiempo de simulación en alguna cantidad "real", es más probable que algo esté mal.

4.7.10 Síntesis

Como mencionamos al principio de esta sección, VHDL fue inventado originalmente como un lenguaje de simulación y descripción de circuitos lógicos y posteriormente fue adaptado para síntesis. El lenguaje tiene muchas características y construcciones que no pueden ser sintetizadas. Sin embargo, el subconjunto del lenguaje y el estilo de programas que hemos presentado en esta sección son generalmente sintetizables por la mayoría de las herramientas.

Aún más, el código que usted escribe puede tener un gran efecto sobre la calidad de los circuitos sintetizados que obtiene. Unos cuantos ejemplos se enumeran a continuación:

- Las estructuras de control "serie" como `if-elsif-elsif-else` pueden dar como resultado una cadena en serie de compuertas lógicas para verificar condiciones. Es mejor utilizar una instrucción `case` o `select` si las condiciones son mutuamente exclusivas.
- Los lazos en procesos son generalmente "desenrollados" o desarrollados para crear múltiples copias de lógica combinacional para ejecutar las instrucciones en el lazo. Si se quiere emplear sólo una copia de la lógica combinacional en una serie de pasos, entonces se tiene que diseñar un circuito secuencial, como discutirá en capítulos posteriores.
- Cuando se utilizan instrucciones condicionales en un proceso, sin establecer un resultado para alguna combinación de entrada, el compilador creará un "latch" para mantener el valor antiguo de una señal que de otra manera puede cambiar. Tales latchs generalmente no son deseados.

Además, algunas características y construcciones del lenguaje pueden simplemente ser no sintetizables, dependiendo de la herramienta. Naturalmente, se tiene que consultar la documentación para averiguar qué está permitido, qué no está permitido y qué está recomendado para una herramienta en particular.

Para el futuro inmediato, los diseñadores digitales que utilizan herramientas de síntesis necesitarán poner atención a sus estilos de codificación, a fin de obtener buenos resultados. Y por el momento, la definición de "buen estilo de codificación" depende de alguna manera tanto de las herramientas de síntesis como de la tecnología destino. Los ejemplos que se mencionan en el resto de este libro, aunque son correctos desde el punto de vista sintáctico y semántico, apenas rascan con dificultad la superficie de los métodos de codificación para diseños extensos basados en HDL. El arte y práctica del diseño a gran escala de hardware basado en HDL está en la etapa inicial de su evolución.

Referencias

Una descripción histórica del desarrollo de Boole de la "ciencia de la lógica" aparece en *The Computer from Pascal to von Neumann* por Herman H. Goldstine (Princeton University Press, 1972). Claude E. Shannon demostró cómo el trabajo de Boole podría ser aplicado a los circuitos lógicos en "A Symbolic-Analysis of Relay and Switching Circuits" *(Trans. AIEE,* Vol. 57, 1938, pp. 713-723).

Aunque el álgebra booleana de dos valores es la base para el álgebra de conmutación, un álgebra booleana no necesita tener solamente dos valores. Las álgebras booleanas con 2^n valores existen para todo entero n; por ejemplo, consúltese *Discrete Mathematical Structures and Their Applications* por Harold S. Stone (SRA, 1973). Tales álgebras se pueden definir formalmente si se aplican los denominados *postulados de Huntington* concebidos por E. V. Huntington en 1907; por ejemplo, véase *Digital Design* por M. Morris Mano (Prentice Hall, 1984). Un desarrollo matemático del álgebra booleana basado en un conjunto más moderno de postulados aparece en *Modern Applied Algebra* por G. Birkhoff y T. C. Bartee (McGraw-Hill, 1970). Nuestro estilo ingenieril, desarrollo "directo" del álgebra de conmutación se sigue de Edward J. McCluskey en su *Introduction to the Theory of Switching Circuits* (McGraw-Hill, 1965) y *Logic Design Principles* (Prentice Hall, 1986).

postulados de Huntington

El teorema de implicantes primos fue demostrado por W. V. Quine en "The Problem of Simplifying Truth Functions" *(Am. Math. Monthly,* Vol. 59, No. 8, 1952, pp. 521-531). De hecho, es posible demostrar un teorema de implicantes primos más general mostrando que existe por lo menos una suma mínima que es una suma de implicantes primos incluso si eliminamos la restricción sobre el número de literales en la definición de "mínima".

Un método gráfico para simplificar funciones booleanas fue propuesto por E. W. Veitch en "A Chart Method for Simplifying Boolean Functions" *(Proc. ACM,* May 1952, pp. 127-133). Su *diagrama de Veitch,* mostrado en la figura 4-53, en realidad reinventó un diagrama propuesto por un arqueólogo inglés, A. Marquand ("On Logical Diagrams for n Terms," *Philosophical Magazine* XII, 1881, pp. 266-270). El diagrama de Veitch o esquema de Marquand hace uso de un orden de conteo binario "natural" para sus ren-

glones y columnas, con el resultado de que algunos renglones y columnas adyacentes difieren en más de un valor, y los términos de producto no siempre cubren las celdas adyacentes. M. Karnaugh demostró cómo arreglar el problema en "A Map Method for Synthesis of Combinational Logic Circuits" *(Trans. AIEE, Comm. and Electron,* Vol. 72, Part 1, noviembre de 1953, pp. 593-599). Por otro lado, George J. Klir, en su libro *Introduction to the Methodology of Switching Circuits* (Van Nostrand, 1972), afirma que el orden de conteo binario es tan bueno, o quizás mejor que el orden del mapa de Karnaugh para minimizar funciones lógicas.

En este punto, el argumento de Karnaugh contra el de Veitch por supuesto que es irrelevante, porque ya no se dibujan diagramas para la minimización de circuitos lógicos. En vez de ello, utilizamos programas de computadora que ejecutan algoritmos de minimización lógica. El primero de tales algoritmos fue descrito por Quine en "A Way to Simplify Truth Functions" *(Am. Math. Monthly,* Vol. 62, No. 9, 1955, pp. 627-631) y modificado por E. J. McCluskey en "Minimization of Boolean Functions" *(Bell Sys. Tech. J.,* Vol. 35, No. 5, Noviembre de 1956, pp. 1417-1444). El algoritmo de Quine-McCluskey se describe completamente en los libros de McCluskey citados con anterioridad.

El libro de 1965 de McCluskey también cubre el algoritmo de consenso iterativo para hallar implicantes primos y demuestra que funciona. El punto de partida para este algoritmo es una expresión de suma de productos, o de manera equivalente, una lista de cubos. Los términos de producto *no necesitan* ser minterms o implicantes primos, pero *pueden* ser alguno o ninguno entre ellos. En otras palabras, los cubos en la lista pueden tener cualquiera y todas las dimensiones, desde 0 hasta n en una función de n variables. Comenzando con la lista de cubos, el algoritmo genera una lista de todos los cubos implicantes primos de la función, sin tener nunca que generar una lista de *minitérminos* completa.

El algoritmo de consenso iterativo fue publicado por primera vez por T. H. Mott, Jr., en "Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants" *(IRE Trans. Electron. Computers,* Vol. EC-9, No. 2, 1960, pp. 245-252). Un algoritmo de consenso generalizado fue publicado por Pierre Tison en "Generalization of Consensus Theory and Application to the Minimization of Boolean Functions" *(IEEE Trans. Electron. Computers,* Vol. EC-16, No. 4, 1967, pp. 446-456). Todos estos algoritmos están descritos por Thomas Downs en *Logic Design with Pascal* (Van Nostrand Reinhold, 1988).

Como explicamos en la sección 4.4.4, el inmenso número de implicantes primos en algunas funciones lógicas hace poco práctico o imposible encontrarlos todos de manera

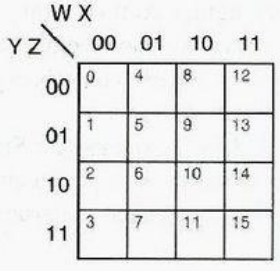


Figura 4-53
Un diagrama de Veitch de 4 variables o diagrama de Marquand.

determinística o seleccionar una cubierta mínima. Sin embargo, usando métodos heurísticos eficientes puede hallarse soluciones que se encuentran cerca del mínimo. El método Espresso-II se describe en *Logic Minimization Algorithms for VLSI Synthesis* por R. K. Brayton, C. McMullen, G. D. Hachtel y A. SangioVanni-Vincentelli (Kluwer Academic Publishers, 1984). Los algoritmos más recientes Espresso-MV y Espresso-EXACT se describen en "Multiple-Valued Minimization for PLA Qptimization" por R. L. Rudell y A. Sangiovanni-Vincentelli (*IEEE Trans. CAD*, Vol. CAD-6, No. 5, 1987, pp. 727-750).

En este capítulo describimos un método de mapa para hallar riesgos estáticos en circuitos AND-OR y OR-AND de dos niveles, pero cualquier circuito combinacional puede ser analizado por riesgos. Tanto en su libro de 1965 como en el de 1986, McCluskey define el *conjunto-0* y los *conjuntos-1* de un circuito y enseña a utilizarlos para encontrar riesgos estáticos. También define los *conjuntos-P* y *conjuntos-S* y muestra cómo pueden emplearse para hallar riesgos dinámicos.

conjunto-0
conjunto-1
conjunto-P
conjunto-S

Muchos aspectos profundos y variados de teoría de conmutación han sido omitidos de este libro, pero han sido apaleados a muerte en otros libros y literatura. Un buen punto de partida para un estudio académico de teoría clásica de conmutación es el libro de Zvi KohaVi, *Switching and Finite Automata Theory*, segunda edición (McGraw-Hill, 1978), el cual incluye material acerca de la teoría de conjuntos, redes simétricas, descomposición funcional, lógica de umbral, detección de fallas y sensibilización de trayectoria. Otra área de gran interés académico (pero poca actividad comercial) es la *lógica multivaluada* no binaria, en la cual cada línea de señal puede tomar más de dos valores. En su libro de 1986, McCluskey proporciona una buena introducción a la lógica multivaluada, explicando sus pros y sus contras y por qué ha habido tan poco desarrollo comercial.

lógica multivaluada

A través de los años, he luchado por hallar una referencia fácilmente accesible y definitiva acerca del lenguaje ABEL, y al fin la encontré: el "Apéndice A" del *Digital Design Using ABEL*, de David Pellerin y Michael Holley (Prentice Hall, 1994). Tiene sentido que esté fuera el trabajo definitivo: ¡Pellerin y Holley inventaron el lenguaje y escribieron el código original del compilador! Pellerin también escribió, con Douglas Taylor como coautor, una muy amena introducción a VHDL, titulada *VHDL Made Easy!* (Prentice Hall, 1997).

VHDL es un "Very Huge Design Language" ("Lenguaje de diseño sumamente enorme") y cubrimos solamente un subconjunto de sus características y capacidades en el libro. Existen muchas opciones del lenguaje (tales como las etiquetas de instrucción) que no utilizamos, pero éstas se estudian en libros tutoriales y de referencia dedicados a VHDL. Uno de los mejores es *The Designer's Guide to VHDL* (Morgan Kaufmann, 1996). Aunque es grande (700 páginas) cubre el lenguaje completo muy sistemáticamente e incluye un apéndice conciso y muy útil acerca de la sintaxis completa del lenguaje tanto de VHDL-87 como de VHDL-93. En este texto empleamos un subconjunto del lenguaje que es compatible con ambos estándares.

Una introducción a VHDL algo breve es *VHDL for Designers* por Stefan Sjoholm y Lennart Lindh (Prentice Hall, 1997). Este libro se recomienda por su ameno enfoque acerca de asuntos de diseño práctico incluyendo síntesis y bancos de prueba. Otro libro

práctico es *VHDL for Programmable Logic*, por Kevin Skahill de Cypress Semiconductor (Addison-Wesley, 1996).

Todos los ejemplos de ABEL y VHDL en este capítulo y a lo largo del libro fueron compilados y en la mayoría de los casos simulados utilizando el software Foundation 1.5 Student Edition de Xilinx, Inc. (San Jose, CA 95124, www.xilinx.com.) El software Foundation integra editor esquemático, editor HDL, compiladores para ABEL, VHDL y Verilog, así como un simulador de Aldec, Inc. (Henderson, NV 89014, www.aldec.com) junto con las herramientas especializadas propiedad de Xilinx para diseño y programación de CPLD y FPGA. También contiene un excelente sistema de ayuda en línea, incluyendo manuales de referencia tanto para ABEL como para VHDL. El software Foundation se distribuye en CD-ROM con algunas ediciones de este libro.

Los paquetes IEEE estándar y VHDL son elementos importantes de cualquier ambiente de diseño VHDL. Una lista de sus definiciones de función y tipo se incluye en un apéndice en algunos textos de VHDL, pero si usted tiene realmente curiosidad acerca de ellos, puede encontrar los archivos fuente completos en el subsistema de biblioteca de cualquier sistema de diseño VHDL, incluyendo el software Foundation.

Discutimos brevemente la prueba de los dispositivos en el contexto de los vectores de prueba de ABEL. Existe un extenso y bien establecido fondo bibliográfico acerca de prueba de dispositivos digitales, y un buen punto de partida para su estudio es el libro de McCluskey de 1986. La generación de un conjunto de vectores de prueba que prueban completamente un circuito extenso tal como un PLD es una tarea que se deja mejor a un programa. Al menos un negocio entero de la compañía se encuentra enfocado a programas que automáticamente crean vectores de prueba para la prueba de PLD (ACUGEN Software, Inc., Nashua, NH 03063, www.acugen.com).

Problemas propuestos

4.1 Utilizando las variables SABELOTODO, DISEÑADOR, FALLA y ESTUDIOS, escriba una expresión booleana que sea 1 para diseñadores con éxito que nunca estudiaron y para sabelotodos que estudiaban todo el tiempo.

4.2 Demuestre los teoremas T2-T5 aplicando la inducción perfecta.

4.3 Demuestre los teoremas T1′-T3′ y T5′ aplicando la inducción perfecta.

4.4 Demuestre los teoremas T6-T9 aplicando la inducción perfecta.

4.5 De acuerdo con el teorema de DeMorgan, el complemento de $X + Y \cdot Z$ es $X' \cdot Y' + Z'$. Ambas funciones todavía son 1 para XYZ = 110. ¿Cómo pueden ser tanto una función como su complemento iguales a 1 para la misma combinación de entrada? Identifique el error.

4.6 Utilice los teoremas del álgebra de conmutación para simplificar cada una de las siguientes funciones lógicas:

(a) $F = W \cdot X \cdot Y \cdot Z \cdot (W \cdot X \cdot Y \cdot Z' + W \cdot X' \cdot Y \cdot Z + W' \cdot X \cdot Y \cdot Z + W \cdot X \cdot Y' \cdot Z)$

(b) $F = A \cdot B + A \cdot B \cdot C' \cdot D + A \cdot B \cdot D \cdot E' + A \cdot B \cdot C' \cdot E + C' \cdot D \cdot E$

(c) $F = M \cdot N \cdot O + Q' \cdot P' \cdot N' + P \cdot R \cdot M + Q' \cdot O \cdot M \cdot P' + M \cdot R$

4.7 Escriba la tabla de verdad para cada una de las siguientes funciones lógicas:

(a) $F = X' \cdot Y + X' \cdot Y' \cdot Z$ (b) $F = W' \cdot X + Y' \cdot Z' + X' \cdot Z$

(c) $F = W + X' \cdot (Y' + Z)$ (d) $F = A \cdot B + B' \cdot C + C' \cdot D + D' \cdot A$

(e) $F = V \cdot W + X' \cdot Y' \cdot Z$ (f) $F = (A' + B' + C \cdot D) \cdot (B + C' + D' \cdot E')$

(g) $F = (W \cdot X)' \cdot (Y' + Z')'$ (h) $F = (((A + B)' + C')' + D)'$

(i) $F = (A' + B + C) \cdot (A + B' + D') \cdot (B + C' + D') \cdot (A + B + C + D)$

4.8 Escriba la tabla de verdad para cada una de las siguientes funciones lógicas:

(a) $F = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z + X \cdot Y' \cdot Z$ (b) $F = M' \cdot N' + M \cdot P + N' \cdot P$

(c) $F = A \cdot B + A \cdot B' \cdot C' + A' \cdot B \cdot C$ (d) $F = A' \cdot B \cdot (C \cdot B \cdot A' + B \cdot C')$

(e) $F = X \cdot Y \cdot (X' \cdot Y \cdot Z + X \cdot Y' \cdot Z + X \cdot Y \cdot Z' + X' \cdot Y' \cdot Z)$ (f) $F = M \cdot N + M' \cdot N' \cdot P'$

(g) $F = (A + A') \cdot B + B \cdot A \cdot C' + C \cdot (A + B') \cdot (A' + B)$ (h) $F = X \cdot Y' + Y \cdot Z + Z' \cdot X$

4.9 Escriba la suma y el producto canónicos para cada una de las siguientes funciones lógicas:

(a) $F = \Sigma_{X,Y}(1,2)$ (b) $F = \prod_{A,B}(0,1,2)$

(c) $F = \Sigma_{A,B,C}(2,4,6,7)$ (d) $F = \prod_{W,X,Y}(0,1,3,4,5)$

(e) $F = X + Y' \cdot Z'$ (f) $F = V' + (W' \cdot X)'$

4.10 Escriba la suma y el producto canónicos para cada una de las siguientes funciones lógicas:

(a) $F = \Sigma_{X,Y,Z}(0,3)$ (b) $F = \prod_{A,B,C}(1,2,4)$

(c) $F = \Sigma_{A,B,C,D}(1,2,5,6)$ (d) $F = \prod_{M,N,P}(0,1,3,6,7)$

(e) $F = X' + Y \cdot Z' + Y \cdot Z'$ (f) $F = A'B + B'C + A$

4.11 Si la suma canónica para una función lógica de n entradas es también una suma mínima, ¿cuántas literales se encuentran en cada término de producto de la suma? ¿Puede haber cualquier otra suma mínima en este caso?

4.12 Proporcione dos razones por las que el costo de inversores no se incluye en la definición de "mínimo" para la minimización lógica.

4.13 Utilice los mapas de Karnaugh para encontrar una expresión de suma de productos mínima para cada una de las siguientes funciones lógicas. Indique las celdas 1 distinguidas en cada mapa.

(a) $F = \Sigma_{X,Y,Z}(1,3,5,6,7)$ (b) $F = \Sigma_{W,X,Y,Z}(1,4,5,6,7,9,14,15)$

(c) $F = \prod_{W,X,Y}(0,1,3,4,5)$ (d) $F = \Sigma_{W,X,Y,Z}(0,2,5,7,8,10,13,15)$

(e) $F = \prod_{A,B,C,D}(1,7,9,13,15)$ (f) $F = \Sigma_{A,B,C,D}(1,4,5,7,12,14,15)$

4.14 Encuentre una expresión de producto de sumas mínima para cada función en el problema propuesto 4.13 utilizando el método de la sección 4.3.6.

4.15 Encuentre una expresión mínima de producto de sumas para la función en cada una de las figuras siguientes y compare su costo con la expresión mínima de producto de sumas previamente hallada: a) figura 4-27; b) figura 4-29; c) figura 4-33.

4.16 Utilizando mapas de Karnaugh, encuentre una expresión de producto de sumas mínima para cada una de las siguientes funciones lógicas. Indique las celdas 1 distinguidas en cada mapa.

(a) $F = \Sigma_{A,B,C}(0,1,2,4)$ (b) $F = \Sigma_{W,X,Y,Z}(1,4,5,6,11,12,13,14)$

(c) $F = \prod_{A,B,C}(1,2,6,7)$ (d) $F = \Sigma_{W,X,Y,Z}(0,1,2,3,7,8,10,11,15)$

(e) $F = \Sigma_{W,X,Y,X}(1,2,4,7,8,11,13,14)$ (f) $F = \prod_{A,B,C,D}(1,3,4,5,6,7,9,12,13,14)$

4.17 Encuentre una expresión de producto de sumas mínima para cada función en el problema propuesto 4.16 utilizando el método de la sección 4.3.6.

4.18 Encuentre la suma completa para las funciones lógicas en el problema 4.16d) y e).

4.19 Empleando mapas de Karnaugh, encuentre una expresión de producto de sumas mínima para cada una de las siguientes funciones lógicas. Indique las celdas 1 distinguidas en cada mapa.

(a) $F = \Sigma_{W,X,Y,Z}(0,1,3,5,14) + d(8,15)$ (b) $F = \Sigma_{W,X,Y,Z}(0,1,2,8,11) + d(3,9,15)$

(c) $F = \Sigma_{A,B,C,D}(1,5,9,14,15) + d(11)$ (d) $F = \Sigma_{A,B,C,D}(1,5,6,7,9,13) + d(4,15)$

(e) $F = \Sigma_{W,X,Y,Z}(3,5,6,7,13) + d(1,2,4,12,15)$

4.20 Repita el problema 4.19, hallando una expresión de producto de sumas mínima para cada función lógica.

4.21 Para cada función lógica en los dos problemas anteriores, determine si la expresión de producto de suma mínima es igual a la expresión de producto de sumas mínima. También compare el costo del circuito para la realización de cada una de las dos expresiones.

4.22 Para cada una de las siguientes expresiones lógicas, encuentre todos los riesgos estáticos en el circuito correspondiente de dos niveles AND-OR u OR-AND, y diseñe un circuito libre de riesgos que realice la misma función lógica.

(a) $F = W \cdot X + W'Y'$ (b) $F = W \cdot X' \cdot Y' + X \cdot Y' \cdot Z + X \cdot Y$

(c) $F = W' \cdot Y + X' \cdot Y' + W \cdot X \cdot Z$ (d) $F = W' \cdot X + Y' \cdot Z + W \cdot X \cdot Y \cdot Z + W \cdot X' \cdot Y \cdot Z'$

(e) $F = (W + X + Y) \cdot (X' + Z')$ (f) $F = (W + Y' + Z') \cdot (W' + X' + Z') \cdot (X' + Y + Z)$

(g) $F = (W + Y + Z') \cdot (W + X' + Y + Z) \cdot (X' + Y') \cdot (X + Z)$

4.23 Escriba un conjunto de vectores de prueba de ABEL para el detector de números primos en la tabla 4-17.

4.24 Escriba un problema VHDL estructural (entidad y arquitectura) para el circuito de alarma en la figura 4-19.

4.25 Repita el problema 4.24, utilizando el estilo de flujo de datos de descripción.

4.26 Repita el problema 4.24, utilice un proceso y un estilo de descripción de comportamiento.

4.27 Escriba un problema VHDL estructural correspondiente al circuito lógico basado en la compuerta NAND de la figura 5-17.

Ejercicios

4.28 Diseñe un circuito lógico de aspecto no trivial con un ciclo de retroalimentación pero que tenga una salida que dependa solamente de su entrada actual.

4.29 Demuestre el teorema de combinación T10 sin utilizar inducción perfecta, pero suponiendo que los teoremas T1-T9 y T1′-T9′ son verdaderos.

4.30 Muestre que el teorema de combinación T10 es de hecho un caso especial del consenso (T11) usado con cobertura (T9).

4.31 Demuestre que $(X + Y') \cdot Y = X \cdot Y$ *sin* emplear inducción perfecta. Se puede suponer que los teoremas T1-T11 y T1′-T11′ son verdaderos.

4.32 Demuestre que $(X + Y) \cdot (X' + Z) = X \cdot Z + X' \cdot Y$, *sin* hacer uso de la inducción perfecta. Se puede suponer que los teoremas T1-T11 y T1′-T11′ son verdaderos.

4.33 Muestre que una compuerta AND de n entradas puede reemplazarse por $(n - 1)$ compuertas AND de 2 entradas. ¿Puede hacerse la misma instrucción para compuertas NAND? Justifique su respuesta.

4.34 ¿Cuántas maneras físicamente diferentes existen para realizar $V \cdot W \cdot X \cdot Y \cdot Z$ utilice cuatro compuertas AND de 2 entradas (4/4 de un 74x08)? Justifique su respuesta.

4.35 Utilice el álgebra de conmutación para demostrar que al unir dos entradas en conjunto de una compuerta AND u OR de $(n + 1)$ entradas proporciona la funcionalidad de una compuerta de n entradas.

4.36 Demuestre los teoremas de DeMorgan (T13 y T13′) usando la inducción finita.

4.37 ¿Cuál símbolo lógico se aproxima con mayor exactitud a la realización interna de una compuerta NOR TTL, la figura 4-4(c) o (d)? ¿Por qué?

4.38 Utilice los teoremas del álgebra de conmutación para volver a escribir la expresión siguiente utilizando tan pocas inversiones como sea posible (se permiten paréntesis complementados):

$$B' \cdot C + A \cdot C \cdot D' + A' \cdot C + E \cdot B' + E \cdot (A + C) \cdot (A' + D')$$

4.39 Demuestre o rechace las instrucciones siguientes:

(a) Sean A y B *variables* del álgebra de conmutación. Entonces $A \cdot B = 0$ y $A + B = 1$ implica que $A = B'$.

(b) Sean X y Y *expresiones* del álgebra de conmutación. Entonces $X \cdot Y = 0$ y $X + Y = 1$ implica que $X = Y'$.

4.40 Demuestre los teoremas de expansión de Shannon. (*Sugerencia:* No se vaya muy lejos: es fácil.)

teoremas generalizados de expansión de Shannon

4.41 Los teoremas de expansión de Shannon pueden generalizarse para "extraer" no solamente una sino i variables de modo que una función lógica pueda ser expresada como una suma o producto de 2^i términos. Establezca los teoremas de expansión generalizados de Shannon.

4.42 Demuestre cómo los teoremas de expansión generalizados de Shannon conducen a las representaciones de suma canónica y producto canónico de funciones lógicas.

compuerta OR Exclusiva (XOR)

4.43 Una *compuerta OR Exclusiva (XOR)* es una compuerta de 2 entradas cuya salida es 1 si y sólo si exactamente una de sus entradas es 1. Escriba una tabla de verdad, expresión de suma de productos y el correspondiente circuito AND-OR para la función OR Exclusiva.

4.44 Desde el punto de vista del álgebra de conmutación, ¿cuál es la función de una compuerta XOR de 2 entradas cuyas entradas se encuentran unidas? ¿Cómo puede diferir el comportamiento de salida de una compuerta XOR real?

4.45 Después de completar el diseño y fabricación de un sistema digital basado en SSI, un diseñador encuentra que se requiere de un inversor más. Sin embargo, las únicas compuertas de sobra en el sistema son una OR de 3 entradas, una AND de 2 entradas y una XOR de 2 entradas. ¿Cómo debería realizar el diseñador la función del inversor sin agregar otro CI?

4.46 Cualquier conjunto de tipos de compuerta lógica que puede realizar cualquier función lógica se conoce como un *conjunto completo* de compuertas lógicas. Por ejemplo, compuertas AND de 2 entradas, compuertas OR de 2 entradas e inversores son un conjunto completo, porque cualquier función lógica puede expresarse como un producto de sumas de variables y sus complementos, y las compuertas AND y OR con cualquier número de entradas pueden hacerse a partir de compuertas de 2 entradas. ¿Las compuertas NAND de 2 entradas forman un conjunto completo de compuertas lógicas? Demuestre su respuesta.

conjunto completo

4.47 ¿Las compuertas NOR de 2 entradas forman un conjunto completo de compuertas lógicas? Demuestre su respuesta.

4.48 ¿Las compuertas XOR de 2 entradas forman un conjunto completo de compuertas lógicas? Demuestre su respuesta.

4.49 Defina una compuerta de dos entradas, aparte de NAND, NOR o XOR, que forme un conjunto completo de compuertas lógicas si se permiten las entradas constantes 0 y 1. Demuestre su respuesta.

4.50 Algunas personas piensan que existen *cuatro* funciones lógicas básicas, AND, OR, NOT y *BUT*. La figura X4.50 es un símbolo posible para una *compuerta* BUT de 4 entradas y 2 salidas. Invente una función útil y no trivial para que la compuerta BUT la realice. La función tendría algo que ver con el nombre (BUT, "pero"). Tenga en cuenta que, debido a la simetría del símbolo, la función debería ser simétrica con respecto a las entradas A y B de cada sección y con respecto a las secciones 1 y 2. Detalle su función de BUT y escriba su tabla de verdad.

BUT
compuerta BUT

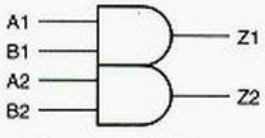


Figura X4.50

4.51 Escriba expresiones lógicas para las salidas Z1 y Z2 de la compuerta BUT que usted diseñó en el ejercicio anterior, y dibuje un diagrama lógico correspondiente utilizando compuertas AND, compuertas OR e inversores.

4.52 La mayoría de los estudiantes no tienen problema al emplear el teorema T8 para "expandir" la multiplicación en expresiones lógicas, pero muchos desarrollan un bloqueo mental si intentan utilizar el teorema T8′ para "expandir" la suma de una expresión lógica. ¿Cómo puede usarse la dualidad para superar este problema?

4.53 ¿Cuántas funciones lógicas diferentes de n variables existen?

4.54 ¿Cuántas funciones lógicas diferentes de 2 variables F(X,Y) existen? Escriba una expresión algebraica simplificada para cada una de ellas.

4.55 Una *función lógica autodual* es una función F tal que $F = F^D$. ¿Cuáles de las siguientes funciones son autoduales? (El símbolo ⊕ denota la operación OR Exclusiva (XOR).)

función lógica autodual
⊕

(a) $F = X$

(b) $F = \Sigma_{X,Y,Z}(0,3,5,6)$

(c) $F = X \cdot Y' + X' \cdot Y$

(d) $F = W \cdot (X \oplus Y \oplus Z) + W' \cdot (X \oplus Y \oplus Z)'$

(e) Una función F de 7 variables tal que F = 1 si y sólo si 4 o más de las variables son 1

(f) Una función F de 10 variables tal que F = 1 si y sólo si 5 o más de las variables son 1

4.56 ¿Cuántas funciones lógicas autoduales de n variables de entrada existen? (*Sugerencia:* Considere la estructura de la tabla de verdad de una función autodual.)

4.57 Demuestre que cualquier función lógica de n entradas $F(X_1,\ldots,X_n)$ que pueda escribirse en la forma $F = X_1 \cdot G(X_2,\ldots,X_n) + X_1' \cdot G^D(X_2,\ldots,X_n)$ es autodual.

4.58 Suponiendo que una compuerta de inversión tiene retardo de propagación de 5 ns, y que una compuerta no inversora tiene un retardo de propagación de 8 ns, compare las velocidades de los circuitos en la figura 4-24 (a), (c) y (d).

4.59 Encuentre las expresiones de producto de sumas mínimo para las funciones lógicas en las figuras 4-27 y 4-29.

4.60 Utilice el álgebra de conmutación para demostrar que las funciones lógicas obtenidas en el ejercicio 4.59 son iguales a las funciones AND-OR obtenidas en las figuras 4-27 y 4-29.

4.61 Determine si las expresiones de producto de sumas obtenidas al "expandir" las sumas mínimas en la figura 4-27 y la 4-29 son mínimas.

4.62 Demuestre que la regla para combinar 2^i celdas 1 en un mapa de Karnaugh es verdadera, utilizando los axiomas y teoremas del álgebra de conmutación.

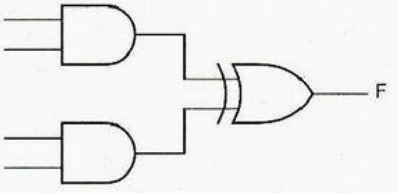
suma no redundante

4.63 Una *suma no redundante* para una función lógica F es una suma de implicantes primos par F de tal modo que si cualquier implicante primo se elimina, la suma ya no es igual a F. Esto suena bastante parecido a una suma mínima, pero una suma no redundante no es necesariamente mínima. Por ejemplo, la suma mínima de la función en la figura 4-35 tiene sólo tres términos de producto, pero hay una suma no redundante con cuatro términos de producto. Encuentre la suma no redundante y dibuje un mapa de la función, encerrando en círculos solamente los implicantes primos en la suma no redundante.

4.64 Encuentre otra función lógica en la sección 4.3 que tenga una o más sumas no redundantes no mínimas, y dibuje su mapa, encerrando en círculos únicamente los implicantes primos en la suma no redundante.

4.65 Dibuje un mapa de Karnaugh y asigne variables a las entradas del circuito AND-XOR en la figura X4.65 de modo que su salida sea $F = \Sigma_{W,X,Y,Z}(6,7,12,13)$. Advierta que la compuerta de salida es una XOR de 2 entradas en lugar de una OR.

Figura X4.65



4.66 Un circuito "comparador" de 3 bits recibe dos números de 3 bits, $P = P_2P_1P_0$ y $Q = Q_2Q_1Q_0$. Diseñe un circuito de suma de productos mínimo que produzca una salida 1 si y sólo si $P > Q$.

4.67 Encuentre expresiones de producto de sumas de salida múltiple mínimas para $F = \Sigma_{X,Y,Z}(0,1,2)$, $G = \Sigma_{X,Y,Z}(1,4,6)$, y $H = \Sigma_{X,Y,Z}(0,1,2,4,6)$.

4.68 Demuestre si la expresión siguiente es una suma mínima o no lo es. Haga esto de la manera más sencilla posible (algebraicamente, sin usar mapas).

$$F = T' \cdot U \cdot V \cdot W \cdot X + T' \cdot U \cdot V' \cdot X \cdot Z + T' \cdot U \cdot W \cdot X \cdot Y' \cdot Z$$

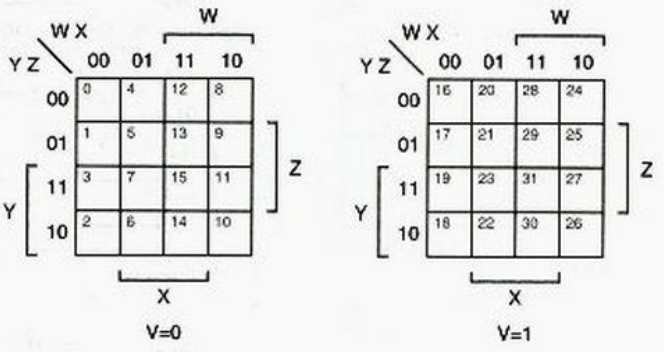


Figura X4.72

4.69 El texto establece que una tabla de verdad o equivalente es el punto de partida para los métodos de minimización combinacionales tradicionales. Un mapa de Karnaugh contiene por sí mismo la misma información que una tabla de verdad. Dada una expresión de suma de productos, es posible escribir los unos correspondientes a cada término de producto directamente en el mapa sin desarrollar una tabla de verdad explícita o lista de minitérminos, y luego proceder con el procedimiento de minimización de mapa. De esta forma, encuentre una expresión de producto de sumas mínimas para cada una de las siguientes funciones lógicas:

(a) $F = X' \cdot Z + X \cdot Y + X \cdot Y' \cdot Z$

(b) $F = A' \cdot C' \cdot D + B' \cdot C \cdot D + A \cdot C' \cdot D + B \cdot C \cdot D$

(c) $F = W \cdot X \cdot Z' + W \cdot X' \cdot Y \cdot Z + X \cdot Z$

(d) $F = (X' + Y') \cdot (W' + X' + Y) \cdot (W' + X + Z)$

(e) $F = A \cdot B \cdot C' \cdot D' + A' \cdot B \cdot C' + A \cdot B \cdot D + A' \cdot C \cdot D + B \cdot C \cdot D'$

4.70 Repita el ejercicio 4.69, encontrando una expresión de producto de sumas mínima para cada función lógica.

4.71 Derive la expresión de producto de sumas mínima para la función del detector de números BCD primos de la figura 4-37. Determine si la expresión es o no algebraicamente igual a la expresión de producto de sumas mínima y explique su respuesta.

4.72 Un mapa de Karnaugh para una función de 5 variables puede dibujarse como se muestra en la figura X4.72. En un mapa así, las celdas que ocupan la misma posición relativa en los submapas V = 0 y V = 1 se consideran adyacentes. (Muchos ejemplos trabajados de mapas de Karnaugh de 5 variables aparecen en las secciones 7.4.4 y 7.4.5). Encuentre una expresión de producto de sumas mínima para cada una de las siguientes funciones utilizando un mapa de 5 variables:

mapa de Karnaugh de 5 variables

(a) $F = \Sigma_{V,W,X,Y,Z}(5,7,13,15,16,20,25,27,29,31)$

(b) $F = \Sigma_{V,W,X,Y,Z}(0,7,8,9,12,13,15,16,22,23,30,31)$

(c) $F = \Sigma_{V,W,X,Y,Z}(0,1,2,3,4,5,10,11,14,20,21,24,25,26,27,28,29,30)$

(d) $F = \Sigma_{V,W,X,Y,Z}(0,2,4,6,7,8,10,11,12,13,14,16,18,19,29,30)$

(e) $F = \Pi_{V,W,X,Y,Z}(4,5,10,12,13,16,17,21,25,26,27,29)$

(f) $F = \Sigma_{V,W,X,Y,Z}(4,6,7,9,11,12,13,14,15,20,22,25,27,28,30) + d(1,5,29,31)$

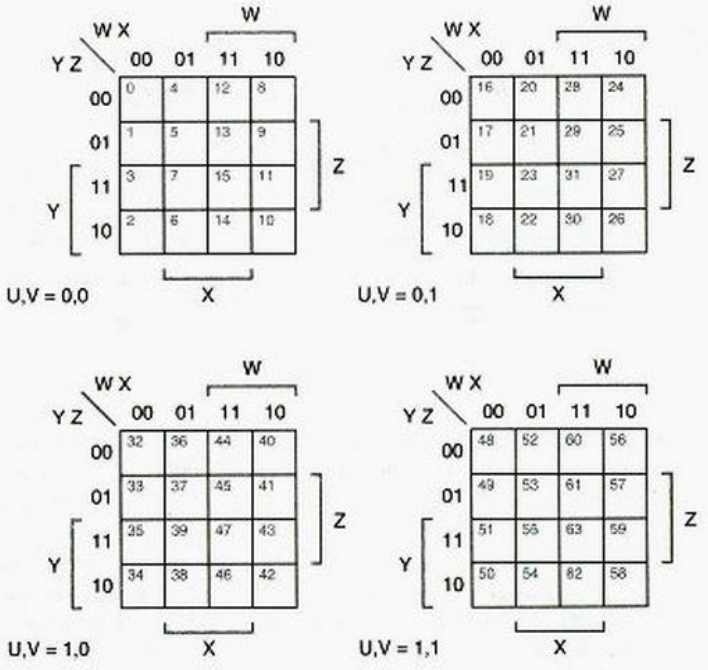


Figura X4.74

4.73 Repita el ejercicio 4.72, hallando una expresión de producto de sumas mínima para cada función lógica.

mapa de Karnaugh de 6 variables

4.74 Un mapa de Karnaugh para una función de 6 variables puede dibujarse como se muestra en la figura X4.74. En un mapa así, las celdas que ocupan la misma posición relativa en submapas adyacentes se consideran como adyacentes. Minimice las funciones siguientes con la ayuda de los mapas de 6 variables:

(a) $F = \Sigma_{U,V,W,X,Y,Z}(1,5,9,13,21,23,29,31,37,45,53,61)$

(b) $F = \Sigma_{U,V,W,X,Y,Z}(0,4,8,16,24,32,34,36,37,39,40,48,50,56)$

(c) $F = \Sigma_{U,V,W,X,Y,Z}(2,4,5,6,12\text{–}21,28\text{–}31,34,38,50,51,60\text{–}63)$

4.75 Existen $2n$ subcubos m de un cubo n para el valor $m = n - 1$. Muestre sus representaciones de texto y los correspondientes términos de producto. (Se pueden emplear elipses a medida que se requieran, por ejemplo, 1, 2, ..., n.)

4.76 Hay sólo un subcubo m de un cubo n para el valor $m = n$; su representación de texto es xx...xx. Escriba el término de producto correspondiente a este cubo.

4.77 El programa en C de la tabla 4-9 utiliza la memoria de manera ineficiente porque asigna la memoria para un número máximo de cubos en cada nivel, incluso si nunca se hace uso de este máximo. Vuelva a diseñar el programa de manera que los arreglos cubes y used sean arreglos de una dimensión, y cada nivel utilice solamente tantas entradas de arreglo como sean necesarias. (*Sugerencia:* se pueden asignar todavía cubos de manera secuencial, pero manténgase al tanto del punto de partida en el arreglo para cada nivel.)

4.78 Como una función de m, ¿cuántas veces se redescubre un cubo m diferente en la tabla 4-9, pero el cubo se encuentra en el ciclo interno y se descarta? Sugiera algunas maneras para eliminar esta deficiencia.

4.79 El tercer ciclo `for` en la tabla 4-9 intenta combinar todos los cubos m a un nivel dado con todos los otros cubos m en ese nivel. De hecho, solamente se pueden combinar los cubos m con las x's en las mismas posiciones, de modo que es posible reducir el número de iteraciones del ciclo mediante el uso de una estructura de datos más sofisticada. Diseñe una estructura de datos que segregue los cubos para un nivel dado de acuerdo a la posición de sus x's, y determine el tamaño máximo requerido para diversos elementos de la estructura de datos. Vuelva a escribir la tabla 4-9 de acuerdo a esto.

4.80 Haga una estimación de si los ahorros en las iteraciones del ciclo interno conseguidas en el ejercicio 4.80 superan el gasto general de mantener una estructura de datos más compleja. Intente hacer suposiciones razonables acerca de cuántos cubos se distribuyen en cada nivel, e indique cómo sus resultados son afectados por estas suposiciones.

4.81 Optimice la función `Oneone` en la tabla 4-8. Una optimización obvia es abandonar el ciclo pronto, pero existen otras optimizaciones que eliminan completamente el ciclo `for`. Una está basada en consulta de tablas y otra utiliza un cálculo delicado que involucra complementación, aplicación de OR Exclusivo y adición.

4.82 Extienda el programa C en la tabla 4-9 para manipular condiciones "sin importancia". Proporcione otra estructura de datos, `dc[MAX_VARS+1][MAX_CUBES]`, que indiquen si un cubo dado contiene solamente elementos "sin importancia", y actualícela a medida que los cubos se leen y generan.

4.83 (*Circuito Hamlet.*) Complete el diagrama de temporización y explique la función del circuito en la figura X4.83. ¿De dónde obtiene su nombre el circuito?

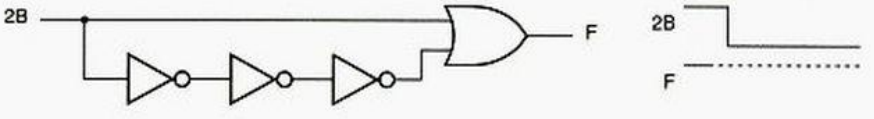


Figura X4.83

4.84 Demuestre que un circuito AND-OR de dos niveles correspondiente a la suma completa de una función lógica siempre está libre de riesgos.

4.85 Encuentre una función lógica de cuatro variables cuya realización de suma de productos mínima no esté libre de riesgos, pero para la cual exista una realización de suma de productos libre de riesgos con menores términos de producto que la suma completa.

4.86 Comenzando con las instrucciones `WHEN` en el programa ABEL de la tabla 4-14, elabore las ecuaciones lógicas para las variables `X4` a `X10` en el programa. Explique cualquier discrepancia entre sus resultados y las ecuaciones en la tabla 4-15.

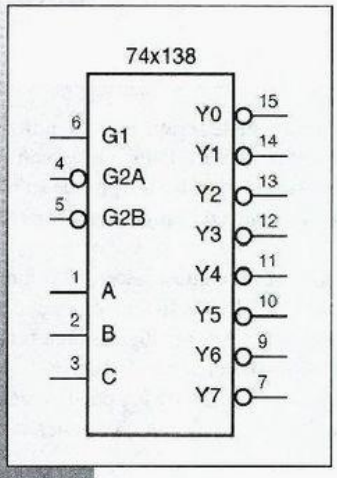
4.87 Dibuje un diagrama de circuito correspondiente a las ecuaciones de suma de productos de dos niveles mínima para el circuito de alarma, como se dan en la tabla 4-12. En cada entrada y salida de inversor, compuerta AND y compuerta OR, escriba un par de números (*t0, t1*), donde *t0* es el número de prueba de la tabla 4-25 que detecta un bloqueo en falla 0 en esa línea, y *t1* es el número de prueba que detecta un bloqueo en falla 1.

4.88 Escriba un programa VHDL en estilo de flujo de datos (entidad y arquitectura) correspondiente al circuito sumador completo en la figura 5-86.

4.89 Utilizando la entidad que usted diseñó en el ejercicio 4.88, escriba un programa VHDL estructural para un sumador cíclico de 4 bits empleando la estructura de la figura 5-87.

4.90 Empleando la entidad que usted definió en el ejercicio 4.88, escriba un programa VHDL estructural para un sumador cíclico de 16 bits a lo largo de las líneas de la figura 5-87. Utilice una instrucción `generate` para crear los 16 sumadores completos y sus conexiones de señal.

4.91 Vuelva a escribir la arquitectura del detector de números primos de la tabla 4-63 utilizando una instrucción `while`.



capítulo 5

Prácticas de diseño lógico combinacional

El capítulo anterior describió los principios teóricos que se utilizan en el diseño lógico combinacional. En este capítulo aplicaremos esos fundamentos y describiremos muchos de los dispositivos, estructuras y métodos que utilizan los ingenieros para resolver problemas prácticos de diseño lógico digital.

Un circuito combinacional práctico puede tener docenas de entradas y salidas y puede requerir cientos, miles, e incluso millones de términos para describir como una suma de productos, y *miles y miles de millones* de renglones para describir en una tabla de verdad. De este modo, la mayoría de los problemas de diseño lógico combinacional reales son demasiado grandes para resolverse mediante la aplicación de la "fuerza bruta" de técnicas teóricas.

Pero, espere, dirá usted, ¿cómo podría cualquier ser humano concebir un circuito lógico así de complejo en primer lugar? La clave es el pensamiento estructurado. Un circuito o sistema complejo se concibe como una colección de subsistemas más pequeños, cada uno de los cuales tiene una descripción mucho más sencilla.

En el diseño lógico combinacional, existen varias estructuras directas: decodificadores, multiplexores, comparadores y demás, que se utilizan con bastante frecuencia como bloques de construcción en sistemas más grandes. Las estructuras más importantes se describen en este capítulo. En general primero describimos cada estructura y posteriormente proporcionamos ejemplos y aplicaciones utilizando componentes de la serie 74, ABEL y VHDL.

Antes de comenzar el anáilisis de estos bloques combinacionales de construcción, necesitamos estudiar varios aspectos importantes. El primer tópico son los estándares de documentación que utilizan los diseñadores digitales para asegurar

LA IMPORTANCIA DE LA SERIE LÓGICA 74

Más adelante, en este mismo capítulo, examinaremos los CI de la serie 74 que se utilizan con bastante frecuencia y realizan funciones lógicas bien estructuradas. Estas partes son importantes bloques de construcción entre las herramientas del diseñador digital, porque su nivel de funcionalidad a menudo coincide con el nivel de pensamiento de un diseñador, cuando se divide un problema extenso en segmentos pequeños.

Aun cuando usted diseñe para los PLD, FPGA o ASIC, la comprensión de las funciones MSI de la serie 74 es importante. En el diseño basado en PLD, las funciones estándar MSI pueden utilizarse como un punto de partida para desarrollar ecuaciones lógicas para funciones más especializadas. Y en el diseño FPGA y ASIC, los bloques de construcción básicos (o "celdas estándar" o "macros"), proporcionados por el fabricante de FPGA o ASIC, pueden realmente ser definidos como funciones MSI de la serie 74, incluso hasta el punto de tener similares números descriptivos.

que sus diseños son correctos, que se pueden fabricar y mantener. Acto seguido, analizamos la temporización del circuito, un elemento crucial para tener éxito en el diseño digital. En tercer lugar, describimos la estructura interna de los PLD combinacionales, los que utilizaremos más adelante como bloques "universales" de construcción.

5.1 Estándares de documentación

Una buena documentación es esencial para lograr un diseño correcto y un mantenimiento eficiente de los sistemas digitales. Además de ser precisa y completa, la documentación debe ser algo instructiva, de modo que un ingeniero de pruebas, técnico de mantenimiento, o inclusive el ingeniero de diseño original (seis meses después de diseñar el circuito) pueda averiguar cómo funciona el sistema con sólo leer la documentación.

Aunque el tipo de documentación depende de la complejidad del sistema y los ambientes de ingeniería y fabricación, un paquete de documentación debería contener (por regla general) al menos los siguientes seis elementos:

especificación de circuito

1. Una *especificación* que describe exactamente lo que se supone debe hacer el circuito o sistema, incluyendo una descripción de todas las entradas y salidas ("interfaces") y las funciones que se van a realizar. Advierta que las "especificaciones" no tienen que especificar *cómo* consigue el sistema sus resultados, sino únicamente lo *que* se supone que son los resultados. Sin embargo, en muchas compañías es una práctica común incorporar también uno o más de los documentos siguientes en las especificaciones para describir cómo funciona el sistema al mismo tiempo.

diagrama de bloques

2. Un *diagrama de bloques* es una descripción pictórica e informal de los principales módulos funcionales del sistema y sus interconexiones básicas.

diagrama esquemático

3. Un *diagrama esquemático* es una especificación formal de los componentes eléctricos del sistema, sus interconexiones y todos los detalles necesarios para construir el sistema, incluyendo tipos de CI, indicadores de referencia y números de terminales. Hemos estado utilizando el término *diagrama lógico* para un dibujo informal que no tiene este nivel de detalle. La mayor parte de los programas de diagramación esquemática tienen la capacidad de generar una *cuenta de materiales (BOM, del inglés bill of materials)* a partir del diagrama esquemático; la cuenta le indica al

diagrama lógico

cuenta de materiales (BOM)

DOCUMENTOS EN LÍNEA

En la actualidad, la documentación profesional que utilizan los ingenieros se mantiene en las intranets (redes internas) corporativas, de modo que es muy útil incluir los URL en las especificaciones del circuito y descripciones de tal modo que las referencias puedan ser fácilmente localizadas. La documentación en línea es tan importante y fidedigna en una compañía que el pie de página en cada página de todas las especificaciones contiene la advertencia "Una versión impresa de este documento es una copia no controlada". Es decir, una copia impresa muy bien podría ser obsoleta.

departamento de compras cuáles componentes eléctricos tienen qué ordenar para construir el sistema.

4. Un *diagrama de temporización* muestra los valores de diversas señales lógicas como función del tiempo, incluyendo los retardos de causa y efecto entre las señales críticas. *diagrama de temporización*

5. Una *descripción de dispositivo de lógica estructurada* que indica la función interna de un dispositivo de lógica programable (PLD, por sus siglas en inglés), arreglo de compuertas programables (FPGA, por sus siglas en inglés) o circuito integrado de aplicación específica (ASIC, por sus siglas en inglés). Normalmente esta descripción se escribe en un lenguaje de descripción de hardware (HDL) tal como ABEL o VHDL, pero puede estar en forma de ecuaciones lógicas, tablas de estado o diagramas de estado. En algunos casos, se puede utilizar un lenguaje de programación convencional, como C, para modelar el funcionamiento de un circuito o para especificar su comportamiento. *descripción de dispositivo de lógica estructurada*

6. Una *descripción del circuito* es un documento de texto narrativo que, en conjunto con la demás documentación, explica cómo funciona internamente el circuito. La descripción del circuito debe incluir cualquier suposición y falla potencial en el diseño y operación del circuito; también debe señalar el uso de cualquier "truco" de diseño que no sea obvio. Una buena descripción de circuito también contiene definiciones de acrónimos y otros términos especializados y tiene referencias a documentos relacionados. *descripción de circuito*

Quizás es probable que usted haya visto diagramas de bloques en muchos contextos. Presentaremos algunas reglas para dibujarlos en la siguiente subsección, y posteriormente en el resto de esta sección nos concentraremos en los diagramas esquemáticos para los circuitos lógicos combinacionales. La sección 5.2.1 presenta los diagramas de temporización. Las descripciones de lógica estructurada en la forma de programas ABEL y VHDL se cubrieron en las secciones 4.6 y 4.7. En la sección 10.1.6 mostraremos cómo se puede utilizar un programa en C para generar el contenido de una memoria de sólo-lectura.

La última área de la documentación, la descripción del circuito, es muy importante en la práctica. De la misma forma que un programador experimentado crea un documento de diseño de programa antes de comenzar a escribir el código, un diseñador lógico con experiencia comienza escribiendo la descripción del circuito antes de realizar el diagrama esquemático. Por desgracia, en ciertas ocasiones la descripción del circuito es el último documento que se genera, y a veces nunca se escribe. Un circuito sin una buena descripción es difícil de depurar, fabricar, probar, mantener modificar y mejorar.

¡NO SE OLVIDE DE ESCRIBIR!

A fin de crear grandes productos, los diseñadores lógicos deben desarrollar sus habilidades de lenguaje y escritura, especialmente en el área de la organización y resumen *lógicos*. Los diseñadores lógicos con mayor éxito (y posteriormente, líderes de proyectos, arquitectos de sistemas y empresarios) son aquellos que comunican sus ideas, propósitos y decisiones de manera eficaz a otros. Aun cuando haya mucha diversión en meter las manos en un laboratorio de diseño digital, ¡no use esto como una excusa para abreviar sus cursos y proyectos de redacción y comunicación!

5.1.1 Diagramas de bloques

diagrama de bloques

Un *diagrama de bloques* muestra las entradas, salidas, módulos funcionales, trayectorias internas de datos e importantes señales de control de un sistema. En general, no debería ser tan detallado como para ocupar más de una página, pero no debe ser demasiado vago o impreciso. Un pequeño diagrama de bloques puede tener de tres a seis bloques, mientras que uno grande puede tener de 10 a 15, dependiendo de la complejidad del sistema. En cualquier caso, el diagrama de bloques debe mostrar los elementos más importantes

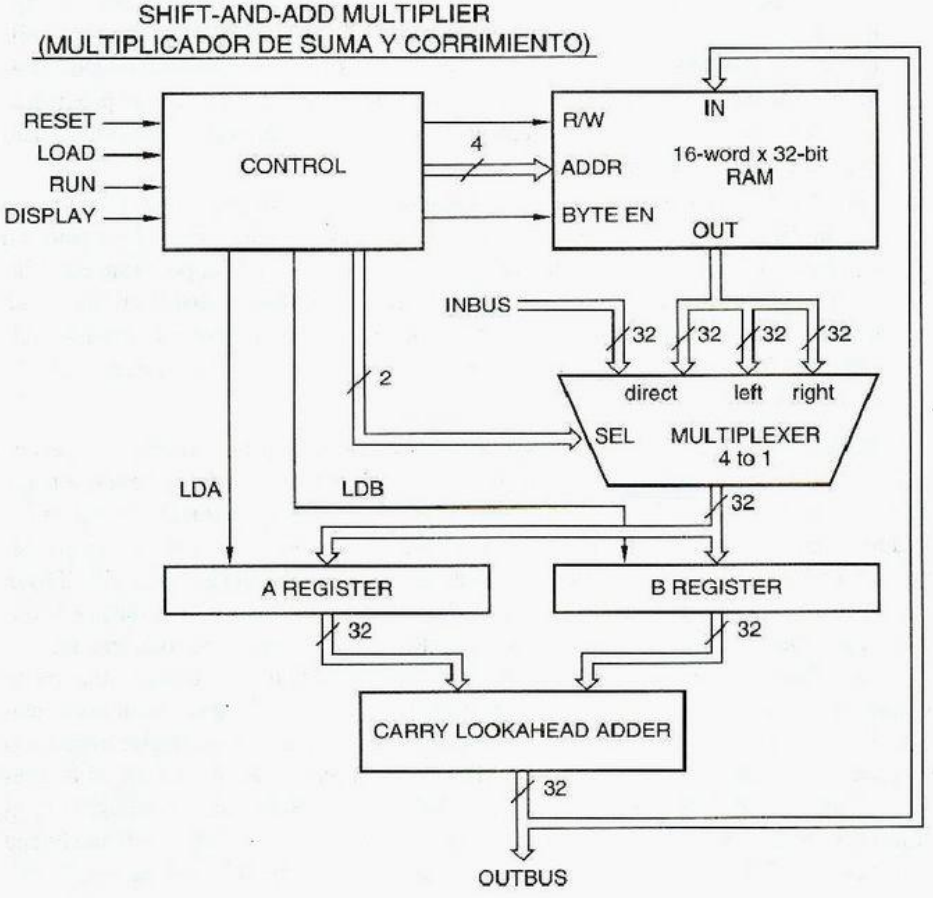


Figura 5-1
Diagrama de bloques para un proyecto de diseño digital.

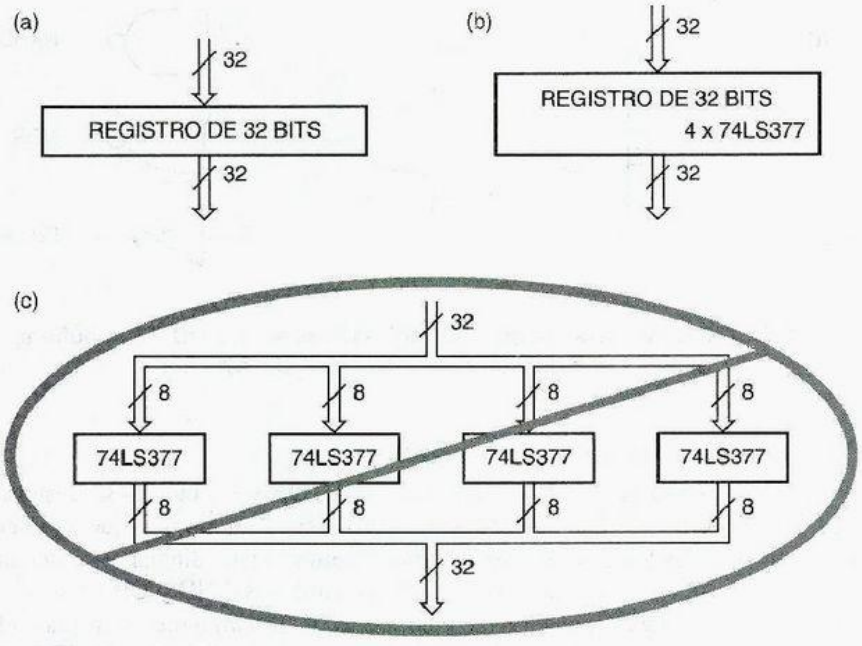


Figura 5-2
Un bloque de registro de 32 bits: (a) realización no especificada; (b) chips especificados; (c) demasiado detalle.

del sistema y cómo funcionan en conjunto. Los sistemas grandes pueden requerir diagramas de bloques adicionales de subsistemas individuales, pero siempre habría un diagrama a "nivel superior" que representa todo el sistema.

La figura 5-1 muestra un diagrama de bloques simple. Cada bloque se identifica con la función del bloque, no con los chips individuales que comprende. Como otro ejemplo, la figura 5-2(a) muestra el símbolo de un diagrama de bloques para un registro de 32 bits. Si el registro va a construirse utilizando cuatro registros 74LS377 de 8 bits, y esta información es importante para alguien que lea el diagrama (por ejemplo, por razones de costo), entonces puede expresarse como se ilustra en (b). Sin embargo, dividir el bloque para mostrar chips individuales como en (c) es incorrecto.

Un *bus* es una colección de dos o más líneas de señal que están relacionadas. En un diagrama de bloques, los buses se dibujan con una línea doble o gruesa. Una diagonal y un número indican cuántas líneas de señal individuales están contenidas en un bus. De manera alternativa, el tamaño puede indicarse en el nombre del bus (por ejemplo, INBUS[31..0] o INBUS[31:0]). Los niveles activos (que se definen más adelante) y las burbujas de inversión pueden o no aparecer en los diagramas de bloques; en la mayoría de los casos, no son importantes para este nivel de detalle. Sin embargo, los buses y las señales de control importantes deberían tener nombres, por lo regular los mismos nombres que aparecen en el diagrama esquemático más detallado.

bus

El flujo de señales de control y datos en un diagrama de bloques debe indicarse claramente. Los diagramas lógicos se dibujan generalmente con las señales fluyendo de izquierda a derecha, pero en los diagramas de bloques este ideal es más difícil de conseguir. Las entradas y salidas pueden hallarse en cualquier lado de un bloque, y la dirección del flujo de la señal puede ser arbitraria. Se utilizan puntas de flecha en los buses y las líneas normales de señal para eliminar la ambigüedad al respecto.

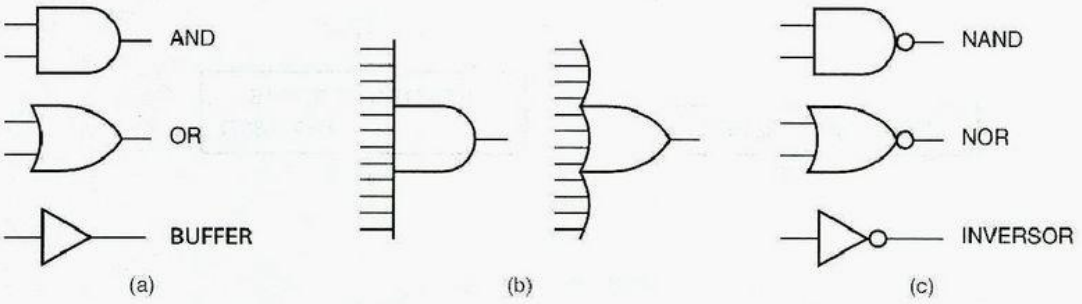


Figura 5-3 Formas para compuertas lógicas básicas: a) AND, OR y buffers; b) expansión de entradas; c) burbujas de inversión.

5.1.2 Símbolos de las compuertas

burbuja de inversión

Las formas de símbolos para las compuertas AND, OR y los buffers se ilustran en la figura 5-3(a). (Recuerde del capítulo 3 que un buffer es un circuito que simplemente convierte las señales lógicas "débiles" en unas "fuertes".) Para dibujar compuertas lógicas con más de algunas entradas, expandimos los símbolos AND y OR como se ilustra en (b). Un círculo pequeño, denominado *burbuja de inversión*, indica la inversión lógica o complementación y se utiliza en los símbolos para las compuertas NAND y NOR e inversores en (c).

Utilizando el teorema de DeMorgan generalizado, podemos manipular las expresiones lógicas para compuertas con salidas complementadas. Por ejemplo, si X y Y son las entradas de una compuerta NAND con salida Z, entonces podemos escribir

$$\begin{aligned} Z &= (X \cdot Y)' \\ &= X' + Y' \end{aligned}$$

Esto da origen a dos símbolos diferentes pero igualmente correctos para una compuerta NAND, como lo demostramos en la figura 4-3 de la página 201. De hecho, esta clase de manipulación puede aplicarse también a compuertas que tienen entradas no complementadas. Por ejemplo, considere las siguientes ecuaciones para una compuerta AND:

$$\begin{aligned} Z &= X \cdot Y \\ &= ((X \cdot Y)')' \\ &= (X' + Y')' \end{aligned}$$

SÍMBOLOS LÓGICOS ESTÁNDAR IEEE

Junto con el American National Standards Institute (ANSI), el Institute of Electrical and Electronic Engineers (IEEE) ha desarrollado un conjunto estándar de símbolos lógicos. La revisión más reciente del estándar es ANSI/IEEE Std 91-1984, *IEEE Standard Graphic Symbols for Logic Functions*, y permite tanto símbolos rectangulares como formas distintivas para compuertas lógicas. Hemos estado utilizando y continuaremos usando los símbolos de formas distintivas en este libro, pero los símbolos de forma rectangular se describen en una guía para los símbolos IEEE en el sitio web www.ddpp.com.

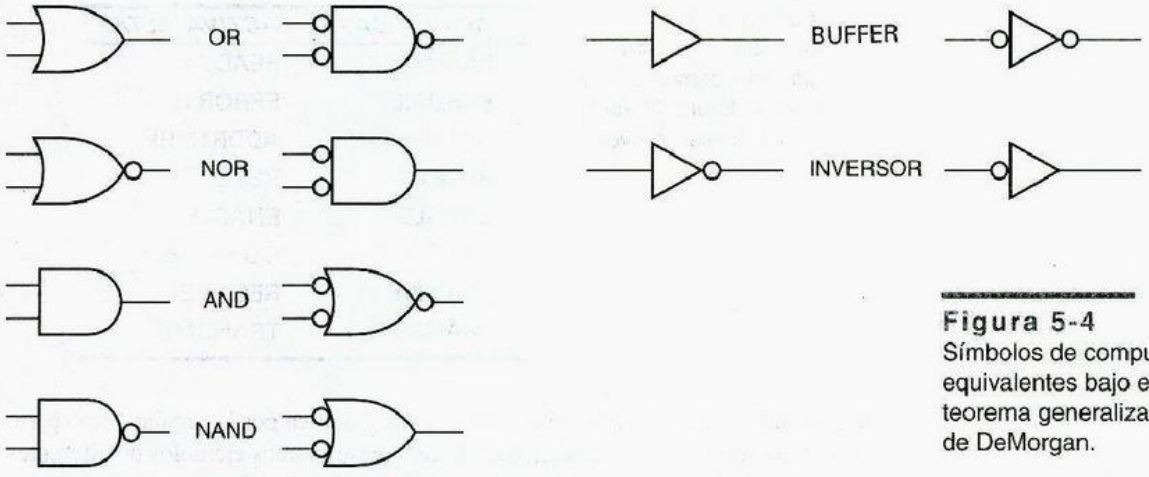


Figura 5-4
Símbolos de compuerta equivalentes bajo el teorema generalizado de DeMorgan.

Así, una compuerta AND puede representarse como una compuerta OR con burbujas de inversión en sus entradas y salidas.

Los símbolos equivalentes para las compuertas estándar que se pueden obtener mediante estas manipulaciones se resumen en la figura 5-4. Aun cuando ambos símbolos en un par representan la misma función lógica, la elección de un símbolo u otro en un diagrama lógico no es arbitraria, al menos no si estamos adheridos a los estándares de una buena documentación. Como veremos en las siguientes tres subsecciones, la selección adecuada de los nombres de señal y símbolos de compuertas pueden facilitar el uso y la comprensión de los diagramas lógicos.

5.1.3 Nombres de señal y niveles activos

Cada señal de entrada y salida en un circuito lógico debería tener una etiqueta alfanumérica descriptiva, el nombre de la señal. La mayor parte de los sistemas de diseño asistidos por computadora para el dibujo de circuitos lógicos también permiten que ciertos caracteres especiales, tales como *, _, y !, se incluyan en los nombres de señal. En los ejemplos de análisis y síntesis del capítulo 4, utilizamos principalmente nombres de señal de un solo carácter, (X, Y, etc.) debido a que los circuitos no eran complejos. Sin embargo, en un sistema real, los nombres de señales bien elegidos transmiten información a cualquiera que lea el diagrama lógico, de la misma manera que los nombres de variables lo hacen en un programa de software. El nombre de una señal indica una acción que es controlada (GO, PAUSE), una condición que se detecta (READY, ERROR) o los datos que se transportan (INBUS[31:0]).

Cada nombre de señal debería tener un *nivel activo* asociado con él. Una señal es *activa-alta* si realiza la acción nombrada o indica la condición nombrada cuando está en nivel ALTO o es 1. (Bajo la convención de lógica positiva, que utilizamos a lo largo de este libro, "ALTO" y "1" son equivalentes.) Una señal es *activa-baja* si realiza la acción nombrada o denota la condición nombrada cuando está en nivel BAJO o es 0. Se dice que una señal es asertiva cuando se encuentra en su nivel activo. Se dice que una señal es de *negación* (o, en ocasiones, *desaserción*) cuando se encuentra en su nivel activo.

nivel activo
activa-alta
activa-baja
negación
desaserción

Tabla 5-1
Cada línea muestra una diferente convención de nomenclatura diferente en los niveles activos.

ACTIVA-BAJA	*ACTIVA-ALTA*
READY-	READY+
ERROR.L	ERROR.H
ADDR15(L)	ADDR15(H)
RESET*	RESET
ENABLE~	ENABLE
~GO	GO
/RECEIVE	RECEIVE
TRANSMIT_L	TRANSMIT

convención de nomenclatura de nivel activo

El nivel activo de cada señal en un circuito se especifica por lo regular como parte de su nombre, de acuerdo a alguna convención. Se muestran unos ejemplos de diferentes *convenciones de identificación de nivel activo* en la tabla 5-1. La elección de una de estas u otras convenciones de identificación de señal es cuestión de preferencia personal, pero con frecuencia se ve restringida por el entorno de la ingeniería. Puesto que la designación del nivel activo es parte del nombre de la señal, la convención de identificación (de la señal) debe ser compatible con los requerimientos de entrada de cualquier herramienta de diseño asistido por computadora que procesará los nombres de señal, tal como los editores de diagramas esquemáticos, compiladores HDL y simuladores. En este texto, utilizaremos la última convención en la tabla: un nombre de señal activa-baja tiene un sufijo de _L, mientras que una señal activa-alta no tiene ese sufijo. El sufijo _L puede leerse como si fuera un prefijo "no".

sufijo _L

nombre de señal
expresión lógica

ecuación lógica

Es extremadamente importante que usted comprenda la diferencia entre nombres de señal, expresiones y ecuaciones. Un *nombre de señal* es solamente un nombre: una etiqueta alfanumérica. Una *expresión lógica* combina nombres de señales que utilizan los operadores del álgebra de conmutación (AND, OR y NOT) como lo explicamos y usamos a lo largo del capítulo 4. Una *ecuación lógica* es la asignación de una expresión lógica para un nombre de señal: describe una función de señal en términos de otras señales.

La distinción entre nombres de señal y expresiones lógicas puede relacionarse con un concepto que se utiliza en los lenguajes de programación de computadoras: el lado izquierdo de una instrucción de asignación contiene un *nombre* de variable, mientras que el lado derecho contiene una *expresión* cuyo valor será dado a la variable nombrada (por ejemplo, `Z = -(X+Y)` en lenguaje C). En un lenguaje de programación, no puede poner una expresión en el lado izquierdo de una instrucción de asignación. En el diseño lógico, no puede emplear una expresión lógica como un nombre de señal.

Las señales lógicas pueden tener nombres como X, READY y GO_L. La "_L" en GO_L es sólo parte del nombre de la señal, como el guión bajo en el nombre de una variable en un programa de C. *No* hay señal cuyo nombre sea READY', ésta es una expresión, puesto que ' es un operador. Sin embargo, pueden existir dos señales denominadas READY y READY_L, de modo que READY_L = READY' durante la operación normal del circuito. Seremos muy cuidadosos en este libro al distinguir entre los nombres de señales, que siempre se encontrarán impresos en negritas, y las expresiones lógicas, que siempre estarán impresas en color gris cuando se escriben cerca de las correspondientes líneas de señal.

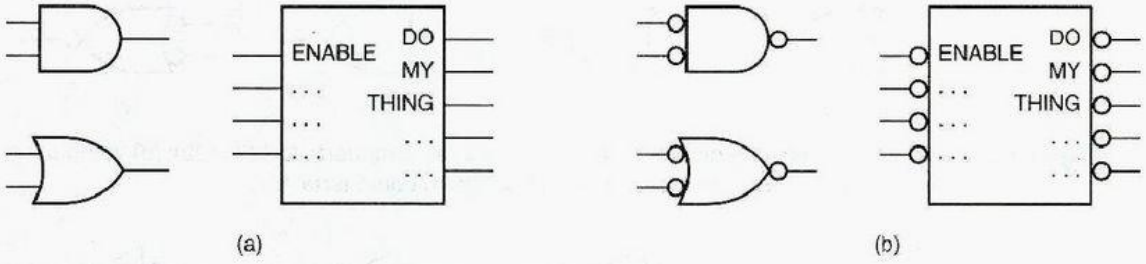


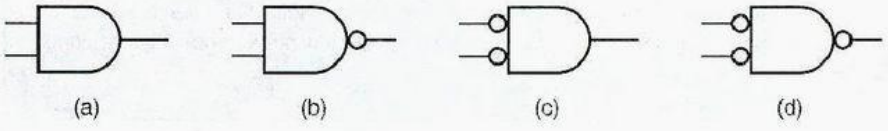
Figura 5-5 Símbolos lógicos; (a) AND, y OR y un elemento lógico de mayor escala; (b) los mismos elementos con entradas y salidas activas-bajas.

5.1.4 Niveles activos para terminales

Cuando dibujamos el esbozo de un símbolo AND u OR, o un rectángulo que representa un elemento lógico de mayor escala, pensamos que la función lógica dada ocurre *dentro* de ese esbozo simbólico. En la figura 5-5(a), mostramos los símbolos lógicos para las compuertas AND, OR y para un elemento de mayor escala con una entrada ENABLE. Las compuertas AND y OR tienen entradas activas altas: requieren 1 en la entrada para validar sus salidas. Igualmente, el elemento de mayor escala tiene una entrada ENABLE activa alta, que debe ser 1 para habilitar el elemento a fin de que cumpla con su función. En (b), mostramos los mismos elementos lógicos con entradas activas bajas y terminales de salida. Exactamente las mismas funciones lógicas se realizan *dentro* de los esquemas simbólicos, pero las burbujas de inversión indican que los 0 deben aplicarse a las terminales de entrada para activar las funciones lógicas, y que las salidas son 0 cuando están "haciendo sus funciones".

De este modo, los niveles activos se pueden asociar con las terminales de entrada y salida de las compuertas y elementos lógicos de mayor escala. Empleamos una burbuja de inversión para indicar una terminal activa baja y la ausencia de una burbuja para indicar una terminal activa alta. Por ejemplo, la compuerta AND en la figura 5-6(a) efectúa el AND lógico, en este caso dos entradas activas altas producen una salida activa alta si ambas entradas son asertivas (1), la salida es asertiva (1). La compuerta NAND en (b) también realiza la función AND, pero produce una salida activa baja. Incluso puede utilizarse una compuerta NOR u OR para realizar la función AND empleando entradas y salidas activas bajas, como se muestra en (c) y (d). Se puede decir que las cuatro compuertas de la figura realizan la misma función: la salida de cada compuerta es asertiva si sus dos entradas son asertivas. La figura 5-7 muestra la misma idea para la función OR: la salida de cada compuerta es asertiva si cualquiera de sus entradas es asertiva.

Figura 5-6 Cuatro formas de obtener una función AND: (a) compuerta AND (74x08); (b) compuerta NAND (74x00); (c) compuerta NOR (74x02); (d) compuerta OR (74x32).



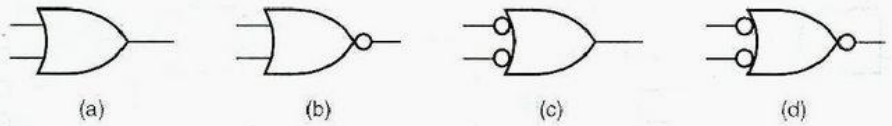


Figura 5-7 Cuatro maneras de obtener una función OR: (a) compuerta OR (74x32); (b) compuerta NOR (74x02); (c) compuerta NAND (74x00); (d) compuerta AND.

Figura 5-8 Símbolos lógicos alternativos: (a, b) inversores; (c, d) buffers no inversores.

En ocasiones se utiliza un buffer no inversor para incrementar el fan out de una señal lógica sin modificar su función. La figura 5-8 ilustra los posibles símbolos lógicos tanto para buffers inversores como para buffers no inversores. En términos de niveles activos, todos los símbolos realizan exactamente la misma función: cada uno activa sus señales de salida si y sólo si su entrada es asertiva.

5.1.5 Diseño de lógica burbuja-burbuja

Los diseñadores experimentados de circuitos lógicos formulan sus circuitos en términos de las funciones lógicas que se efectúan *dentro* de los esquemas simbólicos. Si usted diseña con compuertas discretas o en un HDL como ABEL o VHDL, es más fácil pensar en las señales lógicas y sus interacciones mediante el uso de nombres activo altos. Sin embargo, una vez que está listo para construir su circuito, puede tener que tratar con señales activas bajas debidas a los requerimientos del entorno.

Cuando se diseña con compuertas discretas, ya sea a nivel de tarjeta o de ASIC, un requerimiento clave es con frecuencia la velocidad. Como mostramos en la sección 3.3.6, las compuertas inversoras son más rápidas que las no inversoras, a menudo se obtienen mejoras en el rendimiento cuando se transportan algunas señales en la forma activa baja.

Cuando se diseña con elementos de mayor escala, muchos de ellos pueden hallarse fuera de los chips o fuera de otros componentes existentes que ya tienen algunas entradas y salidas fijas en la forma activa baja. Las razones por las que utilizan señales activas bajas van desde mejoras en el rendimiento hasta años de tradición, pero en cualquier caso usted todavía tiene que lidiar con ello.

¡NOMBRE ESA SEÑAL!

Aunque es absolutamente necesario nombrar solamente las entradas y salidas principales de un circuito, la mayoría de los diseñadores lógicos consideran que nombrar las señales internas es muy útil. Durante la depuración del circuito, es satisfactorio tener un nombre para referirse a una señal interna que se comporta extrañamente. La mayoría de los sistemas de diseño asistido por computadora generan automáticamente etiquetas para las señales sin nombres, pero es preferible un nombre elegido por el usuario que uno generado por la computadora como XSIG1057.

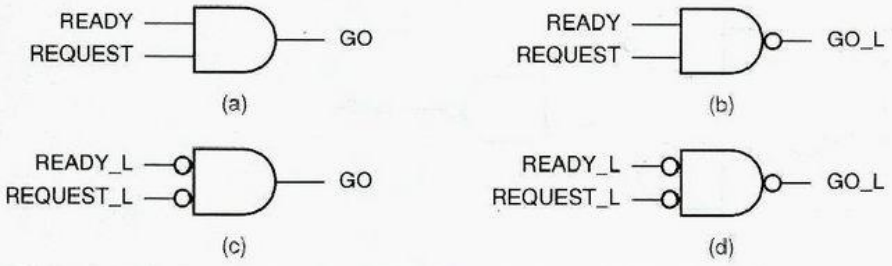


Figura 5-9 Muchas formas de GO: (a) entradas y salidas alto activas; (b) entradas alto activas, salida bajo activa; (c) entradas bajo activas, salida alto activa; (d) entradas y salidas bajo activas.

diseño de lógica burbuja-burbuja

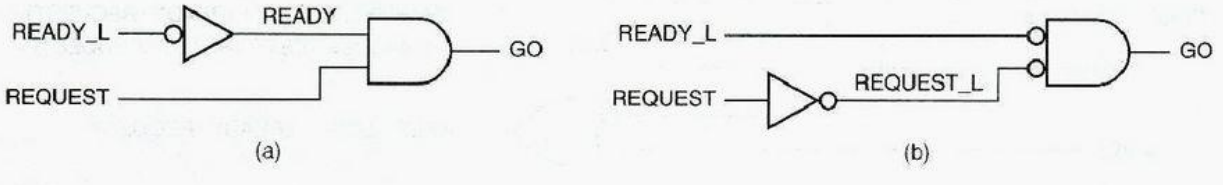
El *diseño de lógica burbuja-burbuja* consiste en la elección de los símbolos lógicos y nombres de señal, incluyendo las denominaciones de nivel activo, que facilitan la comprensión de un circuito lógico. Habitualmente, esto significa seleccionar los nombres de señal, los tipos y símbolos de compuertas, de modo que la mayor parte de las burbujas de inversión se "cancelen" y el diagrama lógico se puede analizar como si todas las señales fueran activas altas.

Por ejemplo, supongamos que necesitamos producir una señal que diga a un dispositivo que "GO" ("arranque") cuando se encuentre "READY" ("listo") y obtengamos "REQUEST" ("solicitud"). Podemos ver claramente que a partir de la instrucción del problema se requiere una función AND; en álgebra de conmutación, escribiríamos GO = READY · REQUEST. Sin embargo, podemos utilizar diferentes compuertas para efectuar la función AND, dependiendo del nivel activo que se requiere para la señal GO, y los niveles activos de las señales disponibles de entrada.

La figura 5-9(a) muestra el caso más simple, donde GO debe ser activa alta y las señales disponibles de entrada también son activas altas; en este caso usamos una compuerta AND. Si, por otro lado, el dispositivo que estamos controlando requiere una señal activa baja GO_L, podemos utilizar una compuerta NAND como se ilustra en (b). Si las señales disponibles de entrada son activas bajas, podemos utilizar una compuerta NOR u OR como se ilustra en (c) y (d).

Los niveles activos de las señales disponibles no siempre coinciden con los niveles activos de las compuertas disponibles. Por ejemplo, supongamos que tenemos dos señales de entrada: READY_L activa baja y REQUEST (activa alta). La figura 5-10 muestra dos formas diferentes para generar GO con la ayuda de un inversor, para obtener el nivel activo necesario para la función AND. Generalmente se prefiere la segunda opción, puesto que las compuertas inversoras, como la NOR, por lo regular son más rápidas que las no inversoras AND. Dibujaremos el inversor de manera diferente en cada caso para hacer que el nivel activo de la salida coincida con el nombre de su señal.

Figura 5-10 Dos formas más de hacer GO, con niveles de entrada mezclados: (a) con una compuerta AND; (b) con una compuerta NOR.



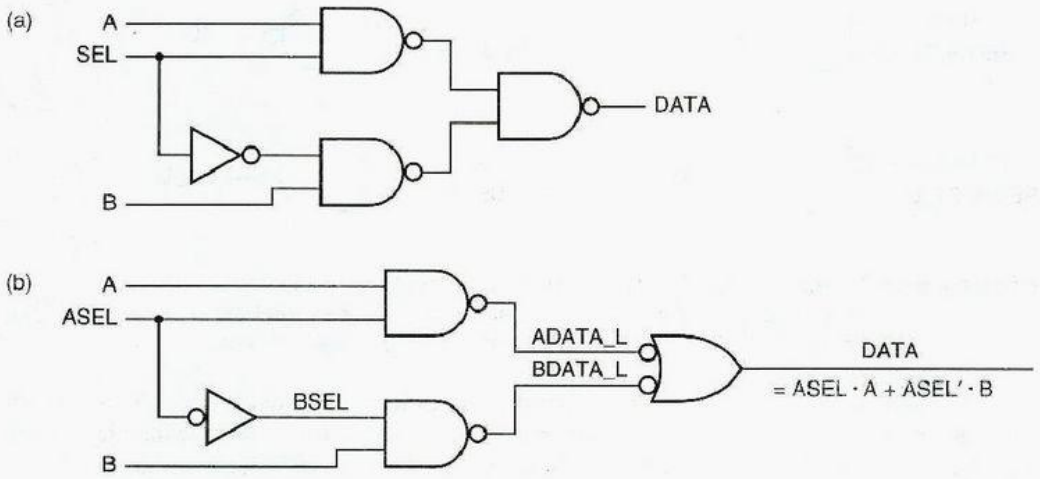
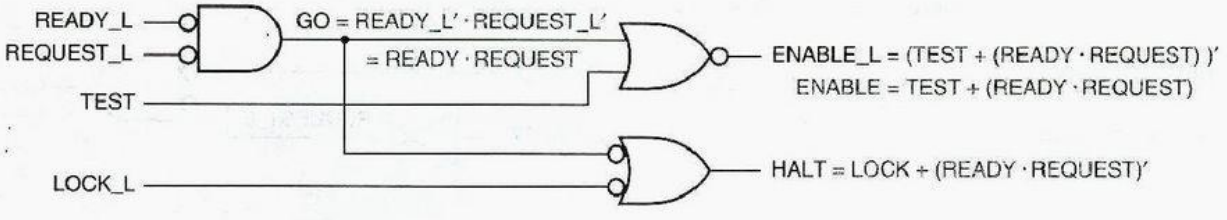


Figura 5-11 Multiplexor de 2 entradas (se supone que usted todavía no sabe qué es esto): (a) diagrama lógico críptico; (b) diagrama lógico apropiado que utiliza denominaciones de nivel activo y símbolos lógicos alternativos.

Para comprender los beneficios que aporta el diseño de lógica burbuja-bubuja, considere el circuito en la figura 5-11(a). ¿Qué es lo que hace? En la sección 4.2 mostramos varias formas de analizar un circuito de esta clase, y ciertamente podíamos obtener una expresión lógica para la salida DATA empleando estas técnicas. Sin embargo, cuando el circuito se vuelve a dibujar como en la figura 5-11(b), la función de salida puede leerse directamente del diagrama lógico, como sigue. La salida DATA es asertiva cuando ADATA_L o BDATA_L son asertivas. Si ASEL es afirmada, entonces ADATA_L es afirmada si y sólo si A es afirmada; es decir, ADATA_L es una copia de A. Si ASEL es negada, BSEL es afirmada y BDATA_L es una copia de B. En otras palabras, DATA es una copia de A si ASEL es afirmada, y DATA es una copia de B si ASEL es negada. Aun cuando existen cinco burbujas de inversión en el diagrama lógico, mentalmente tenemos que realizar sólo una negación para comprender el circuito: que BSEL es asertiva si ASEL no es asertiva.

Si se desea, podemos escribir una expresión algebraica para la salida DATA. Empleamos la técnica de la sección 4.2, propagamos simplemente las expresiones a través de las compuertas hacia la salida. Al hacerlo así, podemos ignorar los pares de burbujas de inversión que se cancelan, y escribir directamente la expresión que se muestra en color gris en la figura.

Figura 5-12 Otro diagrama lógico apropiadamente dibujado.



REGLAS DE DISEÑO DE LÓGICA BURBUJA-BURBUJA

Las siguientes reglas son útiles para efectuar el diseño de lógica burbuja-burbuja:

- El nombre de señal en una salida de dispositivo debería tener el mismo nivel activo que la terminal de salida del dispositivo, es decir, activa baja si el símbolo del dispositivo tiene una burbuja de inversión en la terminal de salida, y activa alta si no es así.
- Si el nivel activo de una señal de entrada es el mismo que el de la terminal de entrada a la cual se encuentra conectada, entonces la función lógica dentro del contorno simbólico se activa cuando la señal es asertiva. Éste es el caso más común en un diagrama lógico.
- Si el nivel activo de una señal de entrada es el opuesto al de la terminal de salida a la cual se encuentra conectada, entonces la función lógica dentro del contorno simbólico es activada cuando la señal es negada. Este caso debería evitarse siempre que fuera posible porque nos obliga a seguir mentalmente la pista de una negación lógica para comprender el circuito.

Otro ejemplo se muestra en la figura 5-12. De la lectura del diagrama lógico, vemos que ENABLE_L es afirmado si READY_L y REQUEST_L se afirman o si TEST se afirma. La salida HALT se afirma si READY_L y REQUEST_L no se afirman, o si LOCK_L se activa. Una vez más, este ejemplo tiene sólo un lugar donde el nivel activo de entrada de la compuerta no coincide con el nivel de la señal de entrada, y esto se refleja en la descripción verbal del circuito.

Podemos, si así lo deseamos, escribir ecuaciones algebraicas para las salidas ENABLE_L y HALT. A medida que se propagan las expresiones a través de las compuertas hacia la salida, obtenemos expresiones como READY_L′ · REQUEST_L′. Sin embargo, podemos utilizar nuestra convención de identificación de nivel activo para simplificar términos como READY_L′. El circuito no contiene señal con el nombre READY, pero si lo hubiera se cumpliría la relación READY = READY_L′ de acuerdo a la convención de identificación. Esto nos permite escribir las ecuaciones ENABLE_L y HALT como se muestra. Complementando ambos lados de la ecuación ENABLE_L, obtenemos una ecuación que describe una salida hipotética ENABLE activa alta en términos de entradas hipotéticas activas altas.

Veremos más ejemplos de diseño con lógica burbuja-burbuja en este capítulo y en los capítulos posteriores, especialmente a medida que empecemos a utilizar elementos lógicos de mayor escala.

5.1.6 Representación de la ubicación de los componentes

Los diagramas lógicos y esquemáticos deberían dibujarse con las compuertas en sus posiciones "normales", con las entradas a la izquierda y las salidas a la derecha. Los símbolos lógicos para elementos lógicos de mayor escala también se dibujan con las entradas a la izquierda y las salidas a la derecha.

La página completa de un diagrama esquemático debería dibujarse con las entradas del sistema a la izquierda y las salidas a la derecha, y el flujo general de las señales debe dirigirse de izquierda a derecha. Si una entrada o salida apareciera a la mitad de la página, deberían extenderse hacia el extremo izquierdo o derecho, respectivamente. De esta manera, un lector puede encontrar todas las entradas y salidas al examinar únicamente los extremos de la página. Todas las trayectorias de señal en la página deben estar conectadas siempre que sea posible; las trayectorias pueden interrumpirse si el dibujo se llena de ellas, pero éstas deben marcarse en ambas direcciones, como se describe más adelante.

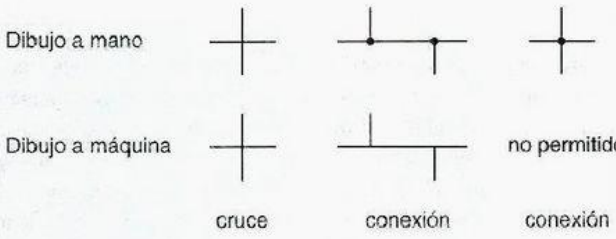
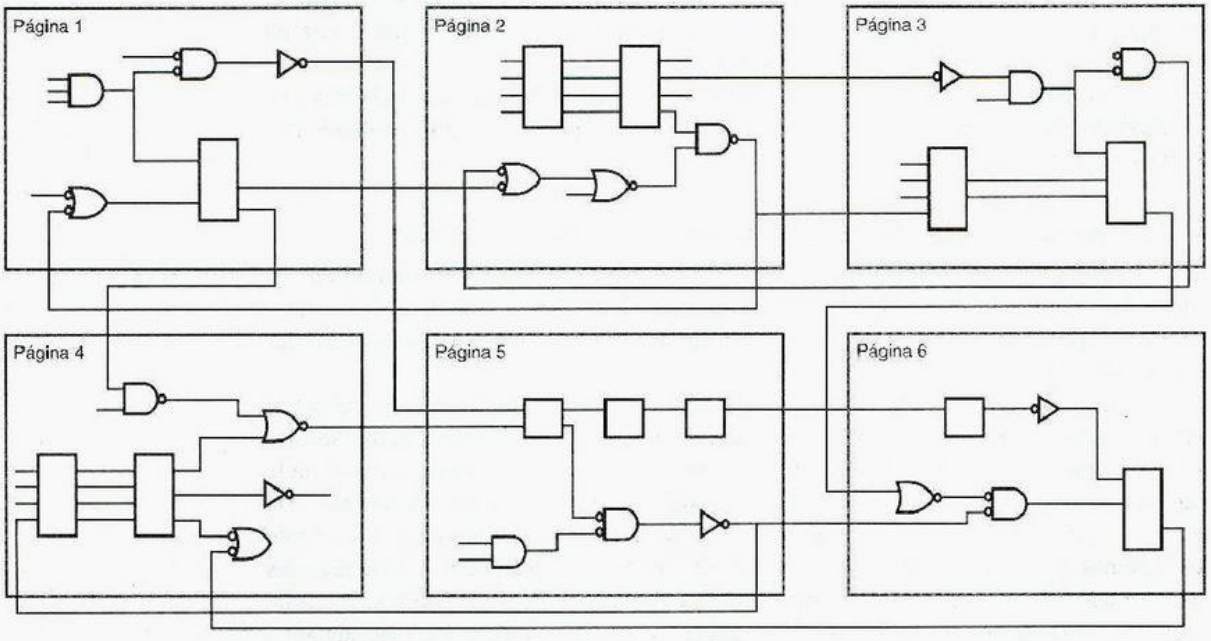


Figura 5-13
Cruces de líneas y conexiones.

En ocasiones los diagramas de bloques se dibujan sin líneas cruzadas para obtener una apariencia más clara, pero esto nunca sucede en los diagramas lógicos. En vez de ello, se permite que las líneas se crucen y las conexiones se indican claramente con un punto. Con todo, algunos sistemas de diseño asistidos por computadora (y algunos diseñadores) no pueden dibujar puntos de conexión legibles. Para distinguir entre líneas que se cruzan y líneas que se conectan, se adoptó la siguiente convención: solamente se permiten las conexiones tipo "T", como se ilustra en la figura 5-13. Ésta es una buena convención a seguir en cualquier caso.

Los diagramas esquemáticos que caben en una sola página son los más sencillos para trabajar. El tamaño más utilizado de la hoja de papel (para un diagrama esquemático) puede ser el tamaño E (34"×44"). Aunque su capacidad de dibujo es grande, un papel así de grande es poco utilizado en el trabajo. El mejor compromiso en cuanto a capacidad para dibujo y factibilidad es el tamaño B (11" ×17"). Las hojas tamaño B se pueden doblar y almacenar con facilidad. Además se pueden consultar rápidamente y si se doblan se pueden guardar en carpetas estándar de 3 argollas; se pueden fotocopiar en la mayor parte de las copiadoras de oficina. Sin tener en cuenta el tamaño del papel, los diagramas esquemáticos

Figura 5-14 Estructura de esquema plano.



se comprenden mejor cuando la página se utiliza en formato "panorámico" (horizontal), es decir, cuando su dimensión más larga está orientada de izquierda a derecha, que es la dirección de flujo de la mayoría de las señales.

Los diagramas esquemáticos que no se ajustan a una solo página deberían dividirse en páginas individuales, en una forma que minimice las conexiones (y la confusión) entre las páginas. También pueden emplear un sistema coordenado, como el de un mapa de carreteras, para señalar las fuentes y destinos de las señales que viajan de una página a otra. Una señal de salida debería tener indicadores para hacer referencia a todos sus destinos, mientras que una señal de entrada debería tener un señalador para hacer referencia a la única fuente. Es decir, una señal de entrada debería ser marcada hacia el lugar donde es generada, no a un lugar en alguna parte a la mitad de una cadena de destinos que utilizan la señal.

marcas de señal

En general, un diagrama esquemático de varias páginas tiene una estructura "plana". Como se ilustra en la figura 5-14, cada página se obtiene a partir del diagrama completo y puede conectarse a cualquier otra página como si todas las páginas estuvieran en una sola hoja grande. Sin embargo, de manera muy parecida a los programas, los diagramas esquemáticos también pueden construirse jerárquicamente, como se ilustra en la figura 5-15. Bajo este enfoque, el diagrama de "nivel superior" es sólo una página simple que puede tomar el lugar de un diagrama de bloques. Típicamente, el diagrama esquemático de nivel superior no contiene compuertas u otros elementos lógicos; solamente muestra los

estructura de diagrama esquemático plano

estructura de diagrama esquemático jerárquico

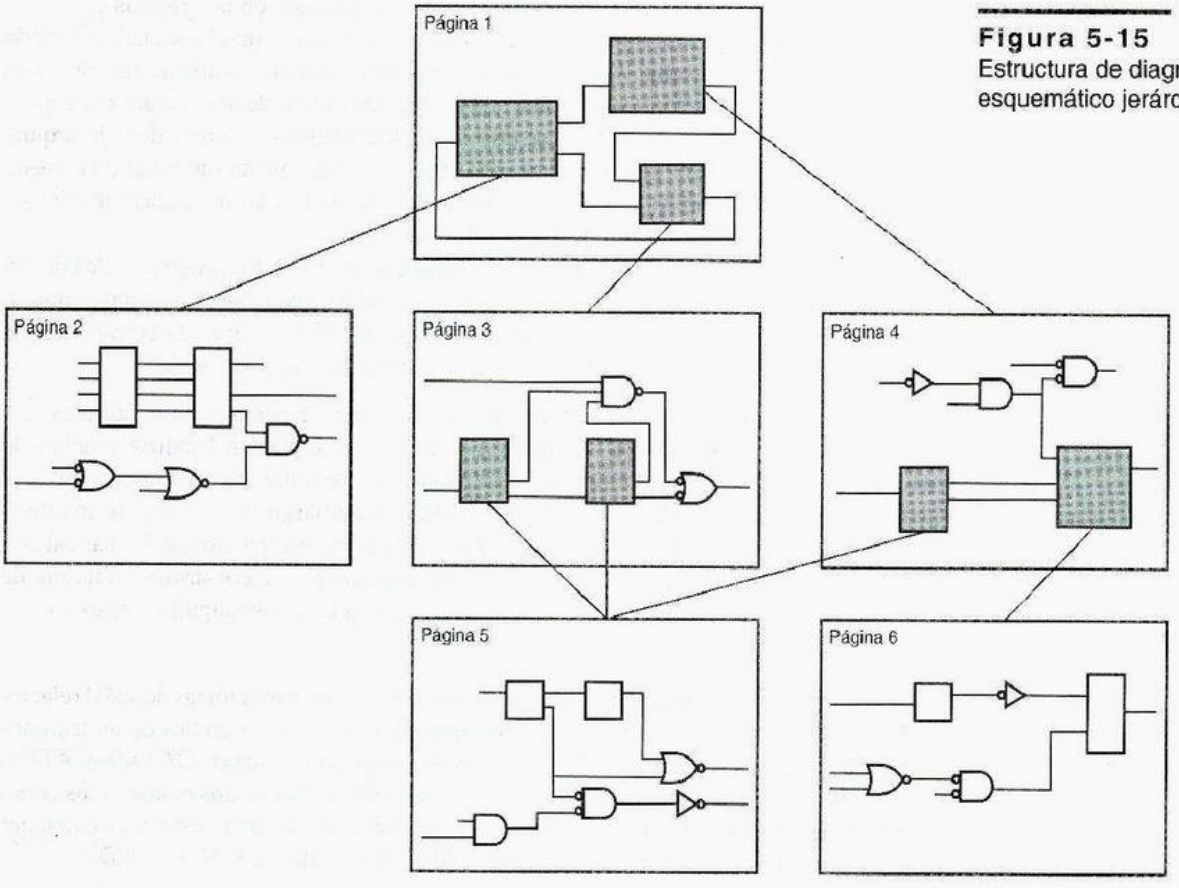


Figura 5-15
Estructura de diagrama esquemático jerárquico.

bloques que corresponden a los subsistemas principales y sus interconexiones. Los bloques o subsistemas se convierten en páginas de nivel inferior, las cuales puedan contener descripciones a nivel de compuerta, o que pueden por sí mismas utilizar bloques definidos en jerarquías de nivel inferior. Si una jerarquía de nivel inferior tiene que utilizarse más de una vez, ésta se puede reutilizar (o "invocar", en el sentido de un programa) varias veces en las páginas de nivel superior.

La mayoría de los sistemas de diseño lógico (asistidos por computadora) soportan tanto a los diagramas esquemáticos planos como a los jerárquicos. Es muy importante una nomenclatura adecuada de las señales en ambos estilos, puesto que hay varios errores comunes que pueden ocurrir:

- Como cualquier otro programa, un programa de captura de diagramas esquemáticos hace lo que usted le indica, no lo que usted quiere decir. Si se utilizan nombres ligeramente diferentes para lo que usted pretende sea la misma señal en páginas diferentes, las señales no se conectarán entre sí.
- A la inversa, si usted utiliza por accidente el mismo nombre para distintas señales en diferentes páginas de un diagrama esquemático plano, muchos programas obedientemente los conectarán entre sí, aun si usted no los ha conectado con una banderita de señalización fuera de la página. (En un diagrama jerárquico, el hecho de reutilizar un nombre en diferentes lugares de la jerarquía generalmente está de bien, porque el programa califica cada nombre con su posición en la jerarquía).
- En un diagrama esquemático y jerárquico debe ponerse mucha atención cuando se identifican las señales de interfaz externa que están en los niveles inferiores de la jerarquía. Éstos son los nombres que aparecerán dentro de los bloques correspondientes a esas páginas, cuando se utilizan en los niveles superiores de la jerarquía. Los nombres de las señales se pueden trasponer con mucha facilidad o se puede utilizar un nombre con el nivel activo equivocado, lo que produciría resultados incorrectos cuando se usa el bloque.
- Aunque no se trata de un problema de identificación, todos los programas de edición para diagramas esquemáticos parecen tener fallas pequeñas, las señales que al parecer están conectadas en realidad no lo están. Si aplicamos la convención en "T" en la figura 5-13 se puede ayudar a minimizar este problema.

Afortunadamente, la mayor parte de los programas de edición para diagramas esquemáticos tienen opciones de verificación de errores, que pueden localizar muchos de estos errores, por ejemplo, cuando se buscan nombres de señal que no tengan entradas, ni salidas, o varias salidas asociadas con ellas. Sin embargo la mayoría de los diseñadores lógicos aprenden la importancia de una cuidadosa esquematización manual con doble verificación, cuando pasan por la amarga experiencia de construir una tarjeta de circuito impreso o un ASIC basados en un diagrama que incluye algún error tonto.

5.1.7 Buses

Como se definió previamente, un bus es una colección de dos o más líneas de señal relacionadas. Por ejemplo, un sistema con microprocesador puede tener un bus de direcciones con 16 líneas, ADDR0–ADDR15, y un bus de datos con 8 líneas, DATA0–DATA7. Los nombres de señal en un bus no están necesariamente relacionados u ordenados como en estos primeros ejemplos. Por ejemplo, un sistema con microprocesador puede tener un bus de control que incluye cinco señales, ALE, MIO, RD_L, WR_L, y RDY.

Los diagramas lógicos utilizan una notación especial para los buses a fin de reducir la cantidad de dibujo y para mejorar la legibilidad. Como se ilustra en la figura 5-16, un bus tiene su propio nombre descriptivo, tal como ADDR[15:0], DATA[7:0], o CONTROL. Un nombre de bus puede utilizar corchetes y el signo de dos puntos para indicar un intervalo. Los buses se dibujan con líneas más gruesas que las líneas comunes de señal. Las señales individuales se introducen o extraen del bus mediante la conexión de una línea de señal ordinaria al bus y se indica el nombre de la señal. Con frecuencia se utiliza también un punto especial de conexión, como en el ejemplo.

Un sistema de diseño asistido por computadora sigue la trayectoria de las señales individuales en un bus. Cuando llega el momento de construir un circuito a partir del diagrama esquemático, las líneas de señal que están en el bus se tratan como si fueran señales independientes.

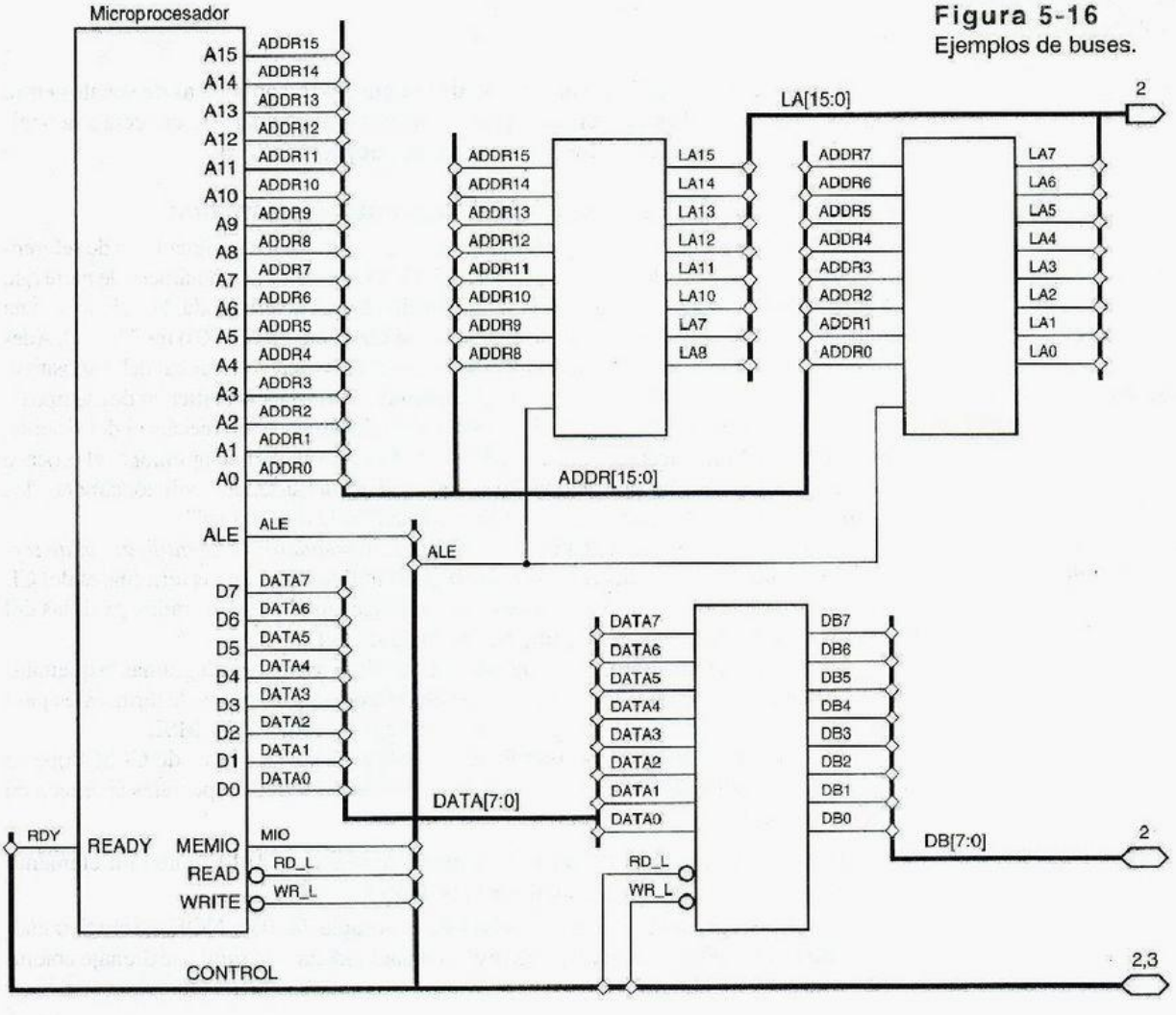


Figura 5-16
Ejemplos de buses.

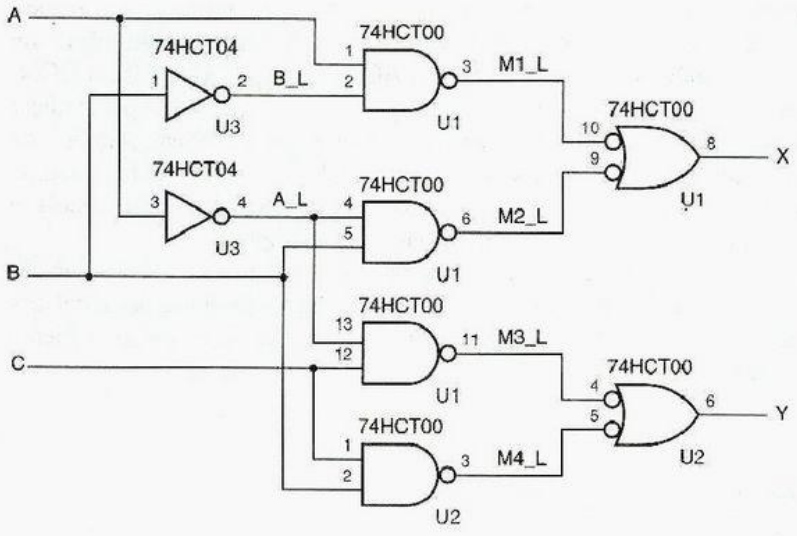


Figura 5-17
Diagrama esquemático para un circuito que utiliza un 74HCT00.

Los símbolos en el extremo derecho de la figura 5-16 son marcas de señales entre páginas. Indican que LA va hacia la página 2, DB es bidireccional y se conecta a la página 2 y CONTROL es bidireccional y se conecta a las páginas 2 y 3.

5.1.8 Información adicional del diagrama esquemático

tipo de CI

Los diagramas esquemáticos completos indican los tipos de CI, los designadores de referencia y números de terminales, como en la figura 5-17. El *tipo de CI* es un número de parte que identifica el circuito integrado que realiza una función lógica determinada. Por ejemplo, una compuerta NAND de 2 entradas se puede identificar como un 74HCT00 o un 74LS00. Además de la función lógica, el tipo de CI identifica la familia lógica y velocidad del dispositivo.

designador de referencia

El *designador de referencia* para un CI identifica una instancia particular de ese tipo de CI que se instala en el sistema. Conjuntamente con la documentación mecánica del sistema, el designador de referencia permite la localización de un CI en particular durante el proceso de ensamble, pruebas y las operaciones de mantenimiento del sistema. Tradicionalmente, los designadores de referencia para CI comienzan con la letra U (de "unidad").

número de identificación de terminal

Una vez que se localiza la posición de un CI, los *números de identificación de terminal* se emplean para localizar las señales lógicas individuales en las terminales del CI. Los números de terminal se escriben cerca de las correspondientes entradas y salidas del símbolo lógico estándar, como se muestra en la figura 5-17.

En el resto de este libro, para que usted se familiarice con los diagramas esquemáticos adecuados, incluiremos los designadores de referencia y números de terminales para todos los ejemplos de circuitos lógicos que utilizan partes tipo SSI y MSI.

La figura 5-18 muestra las terminales de salida de varias clases de CI SSI que se utilizan en los ejemplos de este libro. Algunos elementos gráficos especiales aparecen en ciertos símbolos:

- Los símbolos para el inversor de disparador Schmitt 74x14 tienen un elemento especial en su interior para indicar la histéresis.
- Los símbolos para las compuertas NAND cuádruple 74x03 y NOR Exclusivo cuádruple 74x266 tienen un elemento especial para indicar una salida de drenaje abierto o de colector abierto.

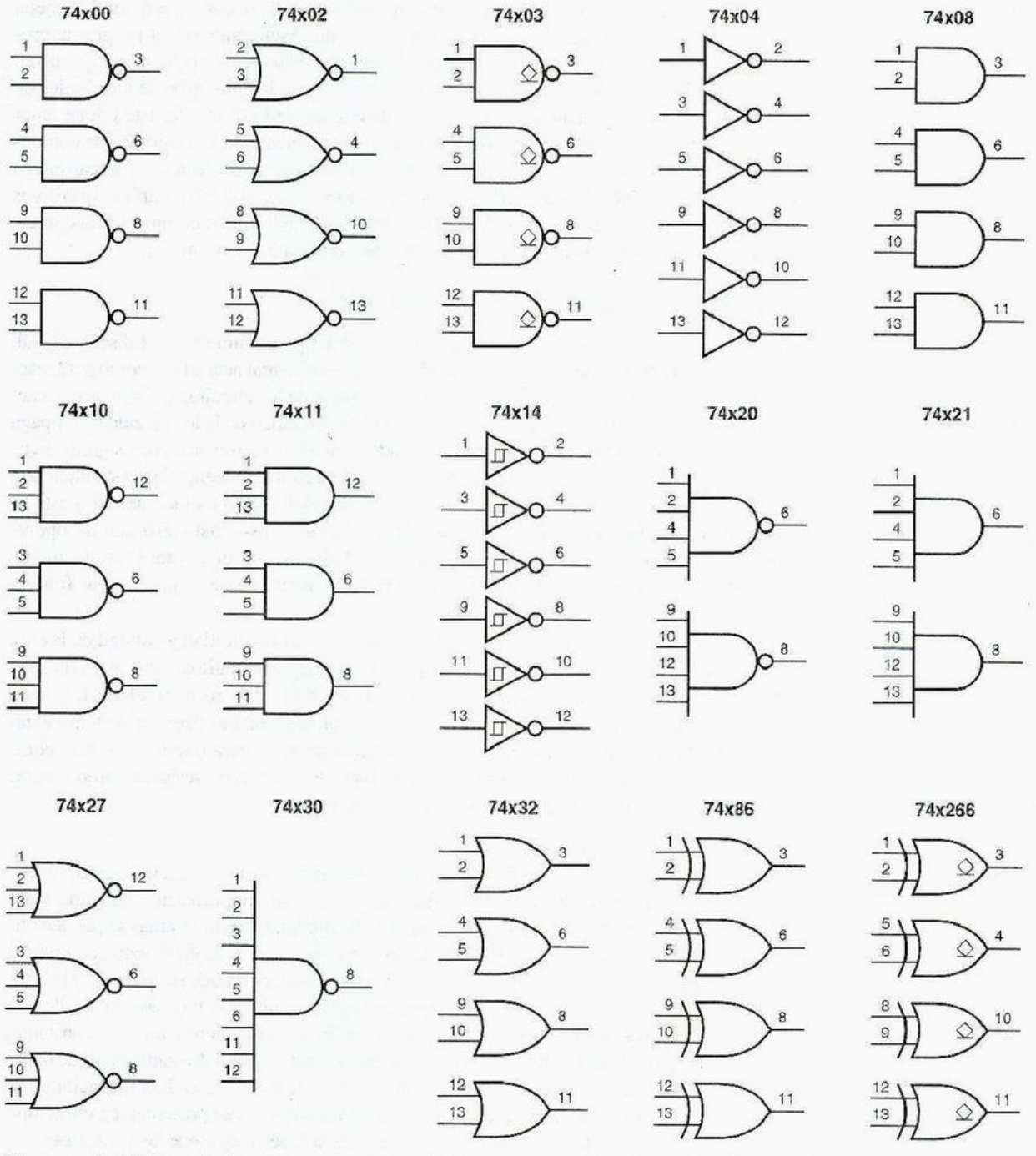


Figura 5-18 Terminales de salida para los CI SSI en encapsulados en doble línea estándar.

Cuando usted prepara un diagrama esquemático para un diseño a nivel de tarjeta, con la ayuda de un programa de captura de diagramas esquemáticos, el programa proporciona en forma automática los números de terminales para los dispositivos que usted seleccione de su biblioteca de componentes. Advierta que los números de terminales de un CI pueden cambiar dependiendo del tipo de encapsulado. Por esto usted debe tener cuidado cuando selecciona la versión correcta del componente de la biblioteca de componentes. La figura 5-18 muestra los números de terminales que se utilizan en un encapsulado de terminales en doble línea (DIP, por sus siglas en inglés), usted utilizaría dispositivos con este encapsulado en un curso de laboratorio de diseño digital o en una tarjeta comercial de circuito impreso de baja densidad (con perforaciones de montaje).

5.2 Temporización del circuito

"El tiempo lo es todo": en inversiones, en la farándula, y sí, también en el diseño digital. Como estudiamos en la sección 3.6, las salidas de un circuito real necesitan tiempo para reaccionar ante los cambios en sus entradas, y la mayor parte de los circuitos y sistemas de la actualidad son tan rápidos, que inclusive el retardo de la velocidad de la luz cuando se propaga una señal de salida hacia una entrada en el otro lado de una tarjeta o de un chip es significativo.

La mayor parte de los sistemas digitales son circuitos secuenciales que funcionan paso a paso bajo el control de una señal periódica de reloj, y la velocidad del reloj está limitada por el tiempo que se necesita —en el peor de los casos— para terminar las operaciones de un paso. De este modo, los diseñadores digitales necesitan estar profundamente conscientes del comportamiento del tiempo, a fin de construir circuitos rápidos que funcionen correctamente en cualquier condición.

En los últimos años se han visto grandes avances en la cantidad y calidad de las herramientas de CAD para analizar la temporización de los circuitos. Aun así, con bastante frecuencia el reto más grande para terminar un diseño a nivel de tarjeta o especialmente un diseño ASIC, consiste en obtener el rendimiento deseado de temporización. En esta sección, comenzaremos con los fundamentos, para que usted pueda comprender lo que hacen las herramientas cuando las utiliza, y logre averiguar cómo reparar sus circuitos cuando su temporización no sea adecuada.

5.2.1 Diagramas de temporización

diagrama de temporización

Un *diagrama de temporización* muestra el comportamiento lógico de las señales en un circuito digital como función del tiempo. Los diagramas de temporización son una parte importante de la documentación de cualquier sistema digital. Estos diagramas se pueden utilizar tanto para explicar las relaciones de tiempo entre las señales de un sistema, como para definir los requerimientos de temporización de las señales externas que se aplican al sistema.

La figura 5-19(a) es el diagrama de bloques de un circuito combinacional sencillo con dos entradas y dos salidas. Suponiendo que la entrada ENB se mantiene a un valor constante, (b) muestra el retardo de las dos salidas con respecto a la entrada GO. En cada forma de onda, la línea superior representa un 1 lógico y la línea inferior, un 0 lógico. Las transiciones de señal se dibujan como líneas inclinadas para recordarnos que no se presentan en un tiempo cero en los circuitos reales (además, las líneas inclinadas lucen mejor que las verticales).

causalidad

retardo

Las flechas se dibujan en ocasiones, especialmente en diagramas complejos de temporización, para mostrar *causalidad*: cuáles transiciones de entrada generan las correspondientes transiciones de salida. En cualquier caso, la información más importante que proporciona un diagrama de temporización es una especificación del *retardo* entre las transiciones.

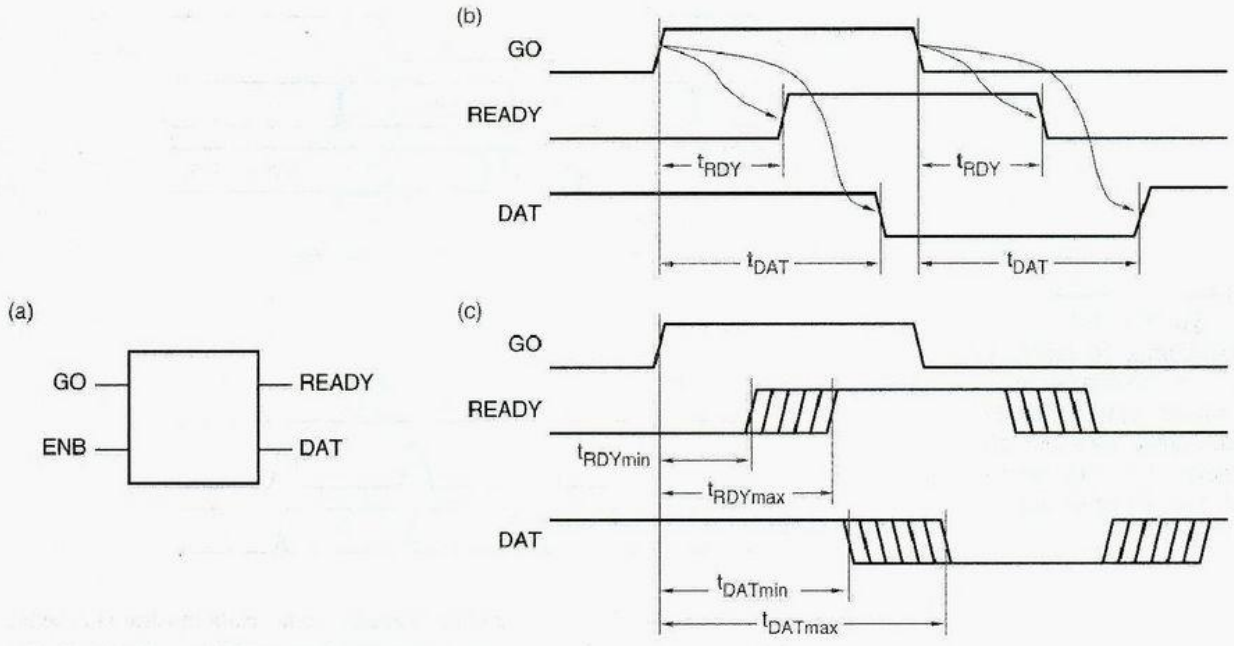


Figura 5-19 Diagramas de temporización para un circuito combinacional: (a) diagrama de bloques del circuito; (b) causalidad y retardo de propagación; (c) retardos mínimo y máximo.

Las diferentes trayectorias a través de un circuito pueden tener distintos retardos. Por ejemplo, la figura 5-19(b) muestra que el retardo de GO hasta READY es más breve que el retardo de GO a DAT. De manera similar, los retardos desde la entrada ENB hasta las salidas pueden variar, y podrían mostrarse en otro diagrama de temporización. Y, como discutiremos más adelante, el retardo a través de cualquier trayectoria determinada puede variar dependiendo de si la salida está cambiando de BAJO a ALTO o de ALTO a BAJO (este fenómeno no se ilustra en la figura).

El retardo en un circuito real se mide normalmente entre los puntos centrales de las transiciones, de manera que los retardos en un diagrama de temporización se marcan en esta forma. Un diagrama de temporización sencillo puede contener varias especificaciones de retardo diferentes. Cada retardo diferente se marca con un identificador distinto, tal como t_{RDY} y t_{DAT} en la figura. En diagramas de temporización más grandes, los identificadores de retardo se numeran para mayor facilidad (por ejemplo, t_1, t_2, ..., t_{42}). En cualquier caso, el diagrama de temporización incluye una *tabla de temporización* que especifica cada cantidad de retardo y las condiciones en las que es válido.

tabla de temporización

Puesto que los tiempos de retardo en los componentes digitales reales pueden variar dependiendo del voltaje, la temperatura y los parámetros de fabricación, el retardo rara vez se especifica como un número simple. En vez de ello, una tabla de temporización puede especificar un intervalo de valores, proporcionando un valor *mínimo, típico* y *máximo* para cada retardo. En ciertas ocasiones la idea de un intervalo de retardos se incluye en el diagrama de temporización al mostrar las transiciones que se presentan en tiempos indeterminados, como en la figura 5-19(c).

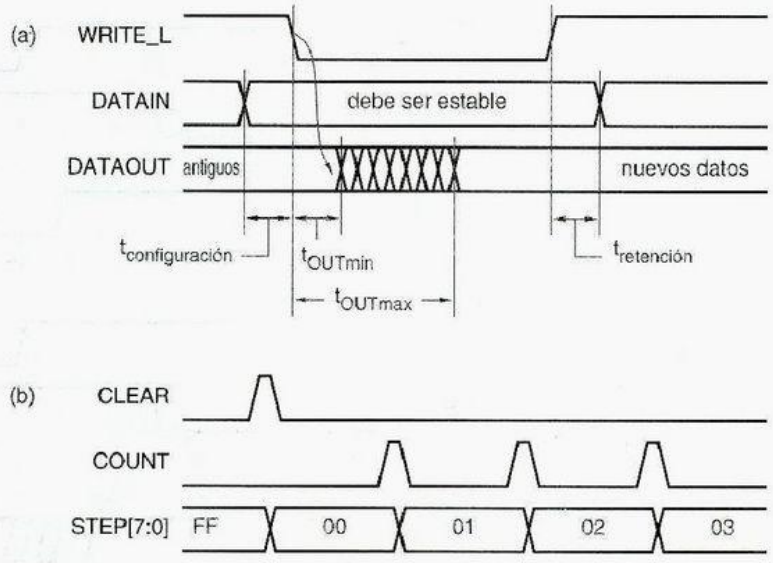


Figura 5-20
Diagramas de temporización para señales de "datos": (a) transiciones determinadas e indeterminadas; (b) secuencia de valores en un bus de 8 bits.

Para algunas señales, el diagrama de temporización no necesita mostrar si la señal cambia desde 0 hasta 1 o desde 1 hasta 0 para un tiempo en particular, solamente que ocurre una transición en ese momento. Cualquier señal que transporte un bit de "datos" tiene esta característica: el valor real del bit de datos varía de acuerdo a las circunstancias pero, sin tener en cuenta el valor, el bit se transfiere, almacena o procesa en un tiempo particular con relación a las señales de "control" del sistema. La figura 5-20(a) es un diagrama de temporización que ilustra este concepto. La señal de "datos" se encuentra normalmente en un valor estacionario de 0 o 1 y las transiciones se presentan solamente en los tiempos indicados. La idea de un tiempo de retardo indeterminado también puede aplicarse a las señales de "datos", como se muestra en la señal DATAOUT.

Muy a menudo en los sistemas digitales, un grupo de señales de datos en un bus se procesan en circuitos idénticos. En este caso, todas las señales en el bus tienen la misma temporización, y pueden representarse con una línea sencilla en el diagrama de temporización y las especificaciones correspondientes en la tabla de temporización. Si los bits del bus se presentan en una combinación particular a un tiempo dado; esto se muestra en ocasiones en el diagrama de temporización utilizando números binarios, octales o hexadecimales, como en la figura 5-20(b).

5.2.2 Retardo de propagación

retardo de propagación

En la sección 3.6.2 definimos formalmente el *retardo de propagación* de una trayectoria de señal como el tiempo que necesita un cambio en la entrada de la trayectoria para producir un cambio en la salida de la trayectoria. Un circuito combinacional con muchas entradas y salidas tiene muchas trayectorias diferentes, y cada una puede tener un retardo de propagación distinto. Además, el retardo de propagación cuando la salida cambia de BAJO a ALTO (t_{pLH}) puede ser diferente del retardo cuando cambia de ALTO a BAJO (t_{pHL}).

El fabricante de un CI de lógica combinacional normalmente especifica todos estos diferentes retardos de propagación, o por lo menos los retardos que serían de interés en

aplicaciones típicas. Un diseñador lógico que combina los CI en un circuito más grande utiliza las especificaciones individuales del dispositivo para analizar la temporización total del circuito. El retardo de una trayectoria a través del circuito es la suma de los retardos a través de las subtrayectorias de los dispositivos individuales.

5.2.3 Especificaciones de temporización

Las especificaciones de temporización para un dispositivo pueden indicar valores mínimos, típicos y máximos para cada trayectoria de retardo de propagación y dirección de transición:

- *Máximo.* Esta especificación es la que utilizan con mayor frecuencia los diseñadores experimentados, puesto que una trayectoria "nunca" tiene un retardo de propagación mayor que el máximo. Sin embargo, la definición de "nunca" varía entre los fabricantes y las familias lógicas. Por ejemplo, los retardos "máximos" de propagación de los dispositivos TTL 74LS y 74S están especificados por $V_{CC} = 5$ V, $T_A = 25°C$ y casi nada de carga capacitiva. Si el voltaje o la temperatura son diferentes, o si la carga capacitiva es superior a 15 pF, el retardo puede alargarse. Por otro lado, un retardo de propagación "máximo" está especificado para los dispositivos 74AC y el 74ACT sobre el intervalo de temperatura y voltaje de operación totales y con una carga capacitiva de 50pF. *retardo máximo*
- *Típico.* Esta especificación es la que más utilizan los diseñadores que no esperan encontrarse disponibles cuando su producto abandone el amigable entorno del laboratorio de ingeniería y se envía a los clientes. El retardo "típico" es el que usted observa en un dispositivo que fue fabricado un "buen día" y funciona bajo condiciones cercanas a las ideales. *retardo típico*
- *Mínimo.* Éste es el retardo de propagación más pequeño que una trayectoria puede exhibir. La mayor parte de los circuitos bien diseñados no dependen de este número; es decir, funcionarán apropiadamente incluso si el retardo es cero. Esto es bueno porque los fabricantes no especifican el retardo mínimo en la mayor parte de las familias lógicas de velocidad moderada, incluyendo las TTL 74LS y 74S. Sin embargo, en las familias de alta velocidad, incluyendo ECL, CMOS 74AC y 74ACT, se especifica un retardo mínimo diferente de cero para ayudar al diseñador a asegurar el cumplimiento de los requerimientos de temporización de los cerrojos y flip-flops, que se analizan en la sección 7.2. *retardo mínimo*

La tabla 5-2 muestra los retardos típicos y máximos de diversas compuertas TTL y CMOS de la serie 74. La tabla 5-3 hace lo mismo para la mayoría de las partes MSI TTL y CMOS que se presentan posteriormente en este capítulo.

¿QUÉ TAN TÍPICO ES LO TÍPICO?

La mayoría de los CI, quizás un 99%, en realidad se fabrican en días "buenos" y muestran retardos cercanos a las especificaciones "típicas". Sin embargo, si usted diseña un sistema que funcione solamente si todos sus 100 CI satisfacen las especificaciones de temporización "típicas", la teoría de la probabilidad sugiere que el 63% ($1 - .99^{100}$) de los sistemas no funcionarán. Pero veamos el siguiente cuadro....

Tabla 5-2 Retardo de propagación en nanosegundos de partes seleccionadas TTL SSI y CMOS de 5 V.

	74HCT		74AHCT				74LS			
	Típico	Máximo	Típico		Máximo		Típico		Máximo	
Número de parte	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}
'00, '10	11	35	5.5	5.5	9.0	9.0	9	10	15	15
'02	9	29	3.4	4.5	8.5	8.5	10	10	15	15
'04	11	35	5.5	5.5	8.5	8.5	9	10	15	15
'08, '11	11	35	5.5	5.5	9.0	9.0	8	10	15	20
'14	16	48	5.5	5.5	9.0	9.0	15	15	22	22
'20	11	35					9	10	15	15
'21	11	35					8	10	15	20
'27	9	29	5.6	5.6	9.0	9.0	10	10	15	15
'30	11	35					8	13	15	20
'32	9	30	5.3	5.3	8.5	8.5	14	14	22	22
'86 (2 niveles)	13	40	5.5	5.5	10	10	12	10	23	17
'86 (3 niveles)	13	40	5.5	5.5	10	10	20	13	30	22

UN COROLARIO DE LA LEY DE MURPHY

La ley de Murphy establece que "si algo puede fallar, fallará". Un corolario a esto es, " si usted quiere que algo salga mal, esto no sucederá".

En el ejemplo mencionado en el cuadro anterior, se puede pensar que se tiene un 63% de oportunidad de detectar los problemas potenciales de temporización en el laboratorio de ingeniería. No obstante, los problemas no se extienden de manera uniforme, puesto que todos los CI de un lote dado tienden a comportarse en forma similar. El corolario de Murphy nos dice que *todos* los prototipos de ingeniería serán construidos con CI de los mismos lotes "buenos". Por lo tanto, todos funcionan bien por un momento, justo el tiempo suficiente para que el sistema llegue a la producción en volumen y para que todos estén satisfechos y se congratulen.

Entonces, de forma desconocida para el departamento de producción, se recibe un lote "lento" de algún tipo de CI y se utiliza en todos los sistemas que se están construyendo, de tal modo que *nada* funciona. Los ingenieros de producción se apresuran intentando analizar el problema (nada fácil, porque el diseñador se marchó hace tiempo y no se tomó la molestia de escribir una descripción del circuito) y, mientras tanto, la compañía pierde considerables cantidades de dinero porque no puede embarcar ningún producto.

Tabla 5-3 Retardo de propagación en nanosegundos de partes seleccionadas CMOS y TTL SSI.

			74HCT		*74AHCT / FCT*		*74LS*			
			Típico	*Máximo*	*Típico*	*Típico*	*Típico*		*Máximo*	
Parte	***De***	***Hasta***	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}
'138	cualquier selección	salida (2)	23	45	8.1 / 5	13 / 9	11	18	20	41
	cualquier selección	salida (3)	23	45	8.1 / 5	13 / 9	21	20	27	39
	$\overline{G2A}$, $\overline{G2B}$	salida	22	42	7.5 / 4	12 / 8	12	20	18	32
	G1	salida	22	42	7.1 / 4	11.5 / 8	14	13	26	38
'139	cualquier selección	salida (2)	14	43	6.5 / 5	10.5 / 9	13	22	20	33
	cualquier selección	salida (3)	14	43	6.5 / 5	10.5 / 9	18	25	29	38
	habilitación	salida	11	43	5.9 / 5	9.5 / 9	16	21	24	32
'151	cualquier selección	Y	17	51	- / 5	- / 9	27	18	43	30
	cualquier selección	$\overline{Y}$	18	54	- / 5	- / 9	14	20	23	32
	cualquier dato	Y	16	48	- / 4	- / 7	20	16	32	26
	cualquier dato	$\overline{Y}$	15	45	- / 4	- / 7	13	12	21	20
	habilitación	Y	12	36	- / 4	- / 7	26	20	42	32
	habilitación	$\overline{Y}$	15	45	- / 4	- / 7	15	18	24	30
'153	cualquier selección	salida	14	43	- / 5	- / 9	19	25	29	38
	cualquier dato	salida	12	43	- / 4	- / 7	10	17	15	26
	habilitación	salida	11	34	- / 4	- / 7	16	21	24	32
'157	selección	salida	15	46	6.8 / 7	11.5 / 10.5	15	18	23	27
	cualquier dato	salida	12	38	5.6 / 4	9.5 / 6	9	9	14	14
	habilitación	salida	12	38	7.1 / 7	12.0 / 10.5	13	14	21	23
'182	cualquier $\overline{Gi}$, $\overline{Pi}$	C1–3	13	41			4.5	4.5	7	7
	cualquier $\overline{Gi}$, $\overline{Pi}$	$\overline{G}$	13	41			5	7	7.5	10.5
	cualquier $\overline{Pi}$	$\overline{P}$	11	35			4.5	6.5	6.5	10
	C0	C1–3	17	50			6.5	7	10	10.5
'280	cualquier entrada	EVEN	18	53	- / 6	- / 10	33	29	50	45
	cualquier entrada	ODD	19	56	- / 6	- / 10	23	31	35	50
'283	C0	cualquier Si	22	66			16	15	24	24
	cualquier Ai, Bi	cualquier Si	21	61			15	15	24	24
	C0	C4	19	58			11	11	17	22
	cualquier Ai, Bi	C4	20	60			11	12	17	17
'381	CIN	cualquier Fi					18	14	27	21
	cualquier Ai, Bi	$\overline{G}$					20	21	30	33
	cualquier Ai, Bi	$\overline{P}$					21	33	23	33
	cualquier Ai, Bi	cualquier Fi					20	15	30	23
	cualquier selección	cualquier Fi					35	34	53	51
	cualquier selección	$\overline{G}$, $\overline{P}$					31	32	47	48
'682	cualquier Pi	$\overline{PEQQ}$	26	69	- / 7	- / 11	13	15	25	25
	cualquier Qi	$\overline{PEQQ}$	26	69	- / 7	- / 11	14	15	25	25
	cualquier Pi	$\overline{PGTQ}$	26	69	- / 9	- / 14	20	15	30	30
	cualquier Qi	$\overline{PGTQ}$	26	69	- / 9	- / 14	21	19	30	30

RETARDOS MÍNIMOS ESTIMADOS

Si el retardo mínimo de un CI no se especifica, un diseñador conservador supone que tiene un retardo mínimo de cero.

Algunos circuitos no funcionan si el retardo de propagación en realidad tiende a cero, pero el costo de modificar un circuito para manipular el caso de retardo cero puede ser poco razonable, en especial si se espera que este caso nunca se presente. Para obtener un diseño que siempre funcione bajo condiciones "razonables", los diseñadores lógicos con frecuencia hacen la estimación de que los CI tienen retardos mínimos de un cuarta a una tercera parte de sus retardos *típicos* publicados.

Todas las entradas de una compuerta SSI tienen el mismo retardo de propagación con respecto a la salida. Advierta que las compuertas TTL habitualmente tienen diferentes retardos para transiciones BAJO a ALTO y ALTO a BAJO (t_{pLH} y t_{pHL}), pero las compuertas CMOS por lo regular no los tienen. Las compuertas CMOS tienen una capacidad de excitación de salida más simétrica, de modo que cualquier diferencia entre ambos casos no tiene importancia.

El retardo desde una transición de entrada a la correspondiente transición de salida depende de la trayectoria interna que toma la señal cambiante, y en circuitos grandes, la trayectoria puede ser diferente para distintas combinaciones de entrada. Por ejemplo, la compuerta XOR 74LS86 de 2 entradas se construye a partir de cuatro compuertas NAND como se muestra en la figura 5-71 , y tiene dos trayectorias con longitudes diferentes desde cualquier entrada hacia la salida. Si una entrada es BAJO, y la otra se modifica, el cambio se propaga a través de dos compuertas NAND y observamos el primer conjunto de retardos que se muestran en la tabla 5-2. Si una entrada es ALTO y la otra se modifica, el cambio se propaga internamente a través de *tres* compuertas NAND, y observamos el segundo conjunto de retardos. Los dispositivos 74LS138 y el 74LS139 presentan un comportamiento en la tabla 5-3. Sin embargo, las partes correspondientes CMOS no muestran estas diferencias; son lo suficientemente pequeñas como para ser ignoradas.

5.2.4 Análisis de temporización

Para analizar con precisión la temporización de un circuito que contiene varios dispositivos SSI y MSI, un diseñador tiene que estudiar su comportamiento lógico con bastante detalle. Por ejemplo, cuando se conectan en serie las compuertas inversoras TTL (NAND, NOR, etc.), un cambio de BAJO a ALTO en la salida de una compuerta produce un cambio de ALTO a BAJO en la siguiente, y así las diferencias entre t_{pLH} y t_{pHL} tienden a promediarse. Por otra parte, cuando las compuertas no inversoras (AND, OR, etc.) se conectan en serie, una transición provoca que todas las salidas cambien en la misma dirección, y de este modo tiende a crecer la brecha entre t_{pLH} y t_{pHL}. Como lector, usted tendrá el privilegio de efectuar esta clase de análisis en los problemas propuestos 5.8-5.13.

El análisis se vuelve más complicado si hay dispositivos MSI en la trayectoria de retardo, o si existen múltiples trayectorias desde una señal de entrada determinada para una señal específica de salida. De esta forma, en circuitos grandes, el análisis de todas las diferentes trayectorias de retardo y las direcciones de transición puede ser muy complejo.

Para permitir un análisis simplificado del "peor de los casos", los diseñadores a menudo hacen uso de una especificación simple del *retardo en el peor de los casos,* que es el máximo de las especificaciones t_{pLH} y t_{pHL}. El retardo del peor de los casos a través de un circuito se calcula entonces como la suma de los retardos en el peor de los casos de los componentes individuales, los cuales son independientes de la dirección de transición y otras condiciones del circuito. Esto puede dar una visión demasiado pesimista del retardo total del circuito, pero ahorra tiempo de diseño y garantiza el funcionamiento del circuito.

retardo en el peor de los casos

5.2.5 Herramientas para análisis de temporización

El uso de sofisticadas herramientas CAD para el diseño lógico facilitan el análisis de temporización. Sus bibliotecas de componentes por lo regular contienen no solamente los símbolos lógicos y los modelos funcionales para diversos elementos lógicos, sino también sus modelos de temporización. Un simulador nos permite aplicar secuencias de entrada y observar cómo y cuándo se producen las salidas. Por lo regular usted puede controlar el uso de los valores de retardo: mínimos, típicos o máximos o alguna combinación de los mismos.

Incluso con un simulador, no se ha librado por completo del atolladero. Por lo general el diseñador suministra las secuencias de entrada para las que el simulador debería producir las salidas. De este modo, se necesitará tener una buena idea de lo que se busca y de cómo estimular su circuito, para producir y observar los retardos del peor de los casos.

Algunos programas de análisis de temporización pueden calcular en forma automática todas las trayectorias de retardo posibles en un circuito e imprimirlas en una lista ordenada, comenzando con la más lenta. No obstante, estos resultados pueden ser demasiado pesimistas, ya que algunas trayectorias pueden no utilizarse durante el funcionamiento normal del circuito; el diseñador debe aplicar su inteligencia para interpretar los resultados de manera apropiada.

5.3 PLD combinacionales

5.3.1 Arreglos de lógica programable

Históricamente, los primeros PLD fueron *arreglos de lógica programable (PLA, programmable logic arrays).* Un PLA es un dispositivo AND-OR combinacional de dos niveles, el cual se puede programar para que realice cualquier expresión lógica de suma de productos, sujeta a las limitaciones de tamaño del dispositivo. Las limitaciones son

arreglo de lógica programable (PLA)

- el número de entradas (n), *entradas*
- el número de salidas (m), y *salidas*
- el número de términos de producto (p). *términos de producto*

Podemos describir un dispositivo de este tipo como "un PLA de $n \times m$ con p términos de producto". En general, p es bastante menor que el número de minitérminos de n variables (2^n). De este modo, un PLA no puede realizar funciones lógicas arbitrarias de n entradas y m salidas; su utilidad está limitada a funciones que puedan expresarse en forma de suma de productos empleando p o menos términos de producto.

Un PLA de $n \times m$ con p términos de producto contiene p compuertas AND de $2n$ entradas y m compuertas OR de p entradas. La figura 5-21 muestra un PLA pequeño con

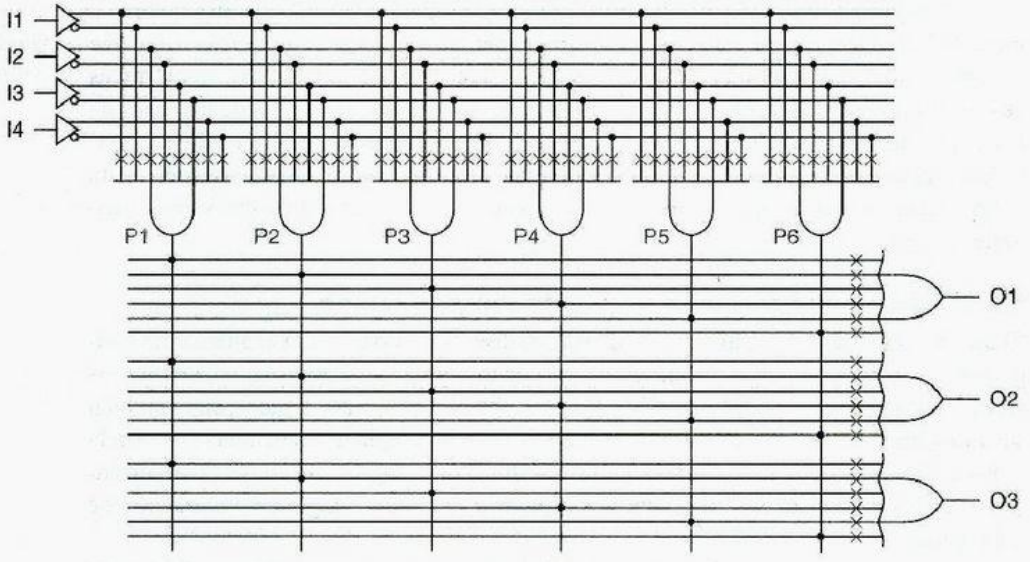


Figura 5-21 Un PLA de 4 × 3 con seis términos de producto.

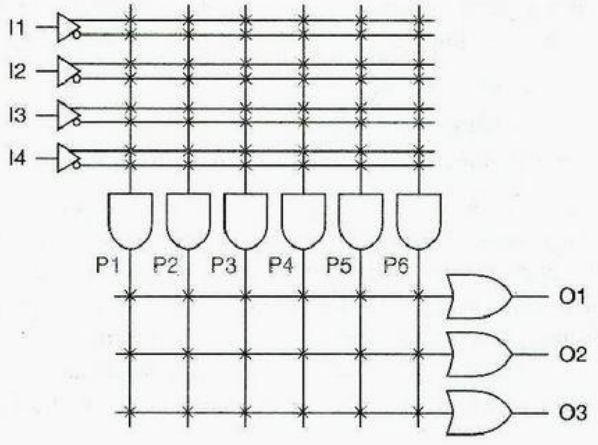
cuatro entradas, seis compuertas AND, tres compuertas OR y tres salidas. Cada entrada se conecta a un buffer que produce tanto una versión verdadera como una complementada de la señal para su uso dentro del arreglo. Las conexiones potenciales en el arreglo están indicadas por X; el dispositivo está programado para establecer solamente las conexiones que realmente son necesarias. Las conexiones necesarias se hacen mediante *fusibles*, que son conexiones de fusible real o celdas de memoria no volátil, dependiendo de la tecnología como se explica en las secciones 5.3.4 y 5.3.5. De esta forma, cada una de las entradas de la compuerta AND puede ser cualquier subconjunto de las señales primarias de entrada y sus complementos. De manera similar, cada una de las entradas de la compuerta OR puede ser cualquier subconjunto de las salidas de la compuerta AND.

Eslabones fusibles PLA

diagrama PLA

Como se ilustra en la figura 5-22, se puede utilizar un diagrama más compacto para representar un PLA. Por otra parte, el formato de este diagrama se parece a la configuración interna real de un chip PLA (por ejemplo, la figura 5-28).

Figura 5-22
Representación compacta de un PLA 4 × 3 con seis términos de producto.



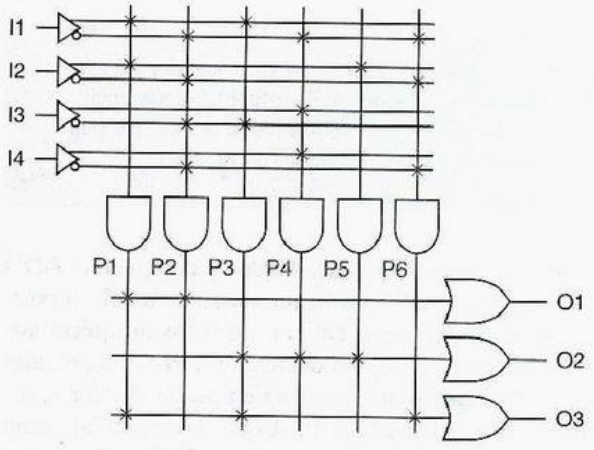


Figura 5-23
Un PLA de 4 × 3 programado con un conjunto de tres ecuaciones lógicas.

El PLA de la figura 5-22 puede efectuar cualquiera de tres funciones lógicas combinacionales de 4 entradas, que se pueden escribir como sumas de productos, utilizando un total de seis o menos términos de productos diferentes, por ejemplo:

$$O1 = I1 \cdot I2 + I1' \cdot I2' \cdot I3' \cdot I4'$$
$$O2 = I1 \cdot I3' + I1' \cdot I3 \cdot I4 + I2$$
$$O3 = I1 \cdot I2 + I1 \cdot I3' + I1' \cdot I2' \cdot I4'$$

Estas ecuaciones tienen un total de ocho términos de productos, pero los primeros dos términos en la ecuación O3 son los mismos que los primeros términos en las ecuaciones O1 y O2. El patrón de conexión programado en la figura 5-23 coincide con estas ecuaciones lógicas.

salidas constantes de PLA

A veces una salida PLA debe programarse para que sea un 1 constante o un 0 constante. No hay problema, como se muestra en la figura 5-24. El término de producto P1 siempre es 1 debido a que su línea de producto no está conectada a ninguna entrada y por lo tanto siempre está en nivel ALTO; este término 1 constante controla la salida O1. Ningún término de producto controla la salida O2, que siempre es 0. Otro método para obtener una salida 0 constante se muestra para O3. El término de producto P2 está conectado a cada variable de entrada y su complemento; por consiguiente, siempre es 0 ($X \cdot X' = 0$).

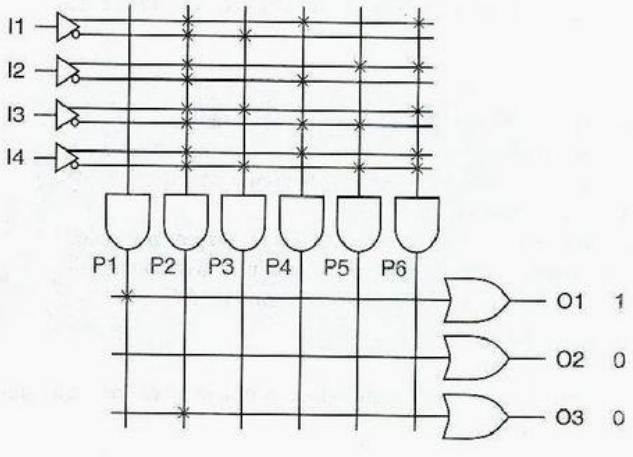


Figura 5-24
Un PLA de 4 × 3 programado para producir salidas constantes 0 y 1.

UNA INTERFERENCIA POCO PROBABLE

Teóricamente, si todas las variables de entrada en la figura 5-24 cambian *simultáneamente*, la salida del término de producto P2 podría tener un breve "glitch" (interferencia) 0-1-0. Esto es poco probable en aplicaciones típicas y es imposible si ocurre que una entrada esté sin utilizar y se encuentre conectada a una señal lógica constante.

Nuestro PLA de ejemplo tiene muy pocas entradas, salidas y compuertas AND (términos de producto) como para ser muy útil. Un PLA de n entradas podría utilizar posiblemente tanto como 2^n términos de producto, para realizar todos los minitérminos de n variables posibles. El número real de términos de producto en los PLA comerciales típicos es bastante menor, del orden de 4 a 16 por salida, sin tener en cuenta el valor de n.

El Signetics 82S100 era un ejemplo típico de los PLA que se comercializaron a mediados de los años 1970. Tenía 16 entradas, 48 compuertas AND y 8 salidas. Así, tenía $2 \times 16 \times 48 = 1536$ fusibles en el arreglo AND y $8 \times 48 = 384$ en el arreglo OR. Los PLA como el 82S100 han sido sustituidos por dispositivos PAL, CPLD y FPGA, pero con bastante frecuencia se sintetizan los PLA personalizados para realizar lógica combinacional compleja en el interior de un ASIC más grande.

5.3.2 Dispositivos lógicos de arreglo programable

dispositivo de lógica de arreglo programable (PAL)

Un caso especial de PLA, y el tipo de PLD que se utiliza con mayor frecuencia en la actualidad, es el *dispositivo lógico de arreglo programable (PAL, programmable array logic)*. A diferencia de un PLA, en el cual tanto el arreglo AND como el OR son programables, un dispositivo PAL tiene un arreglo OR *fijo*.

Los primeros dispositivos PAL* empleaban tecnología bipolar compatible con TTL y se comercializaron a finales de la década de 1970. Las innovaciones clave en los primeros dispositivos PAL, aparte de la introducción de un acrónimo fácil de recordar, fueron el uso de un arreglo OR fijo y terminales bidireccionales de entrada/salida.

PAL16L8

Estas ideas están presentes en el *PAL16L8*, que se muestra en las figuras 5-25 y 5-26, y en una de las estructuras PLD combinacionales que se utilizan con mayor frecuencia en la actualidad. Su arreglo AND programable tiene 64 renglones y 32 columnas, que están identificados para propósitos de programación mediante los pequeños números en la figura, y $64 \times 32 = 2048$ fusibles. Cada una de las 64 compuertas AND del arreglo tiene 32 entradas, que soportan 16 variables y sus complementos; de aquí el "16" en "PAL16L8".

AMIGOS Y ENEMIGOS

PAL es una marca registrada de Advanced Micro Devices, Inc. Como otras marcas registradas, debería emplearse sólo como un adjetivo. Puede utilizarla como sustantivo o sin indicar que es una marca registrada bajo su propio riesgo (yo lo aprendí en una carta que me enviaron los abogados de AMD, en febrero de 1989).

Para eliminar el problema de la marca registrada de AMD, yo sugiero que usted utilice un nombre descriptivo que es más indicativo de la estructura interna del dispositivo: un *FOE ("enemigo", en inglés), de "Fixed-OR Element", elemento de OR fijo.*

*N. Del T. En inglés, el acrónimo PAL no debe confundirse con la palabra "pal" que significa amigo.

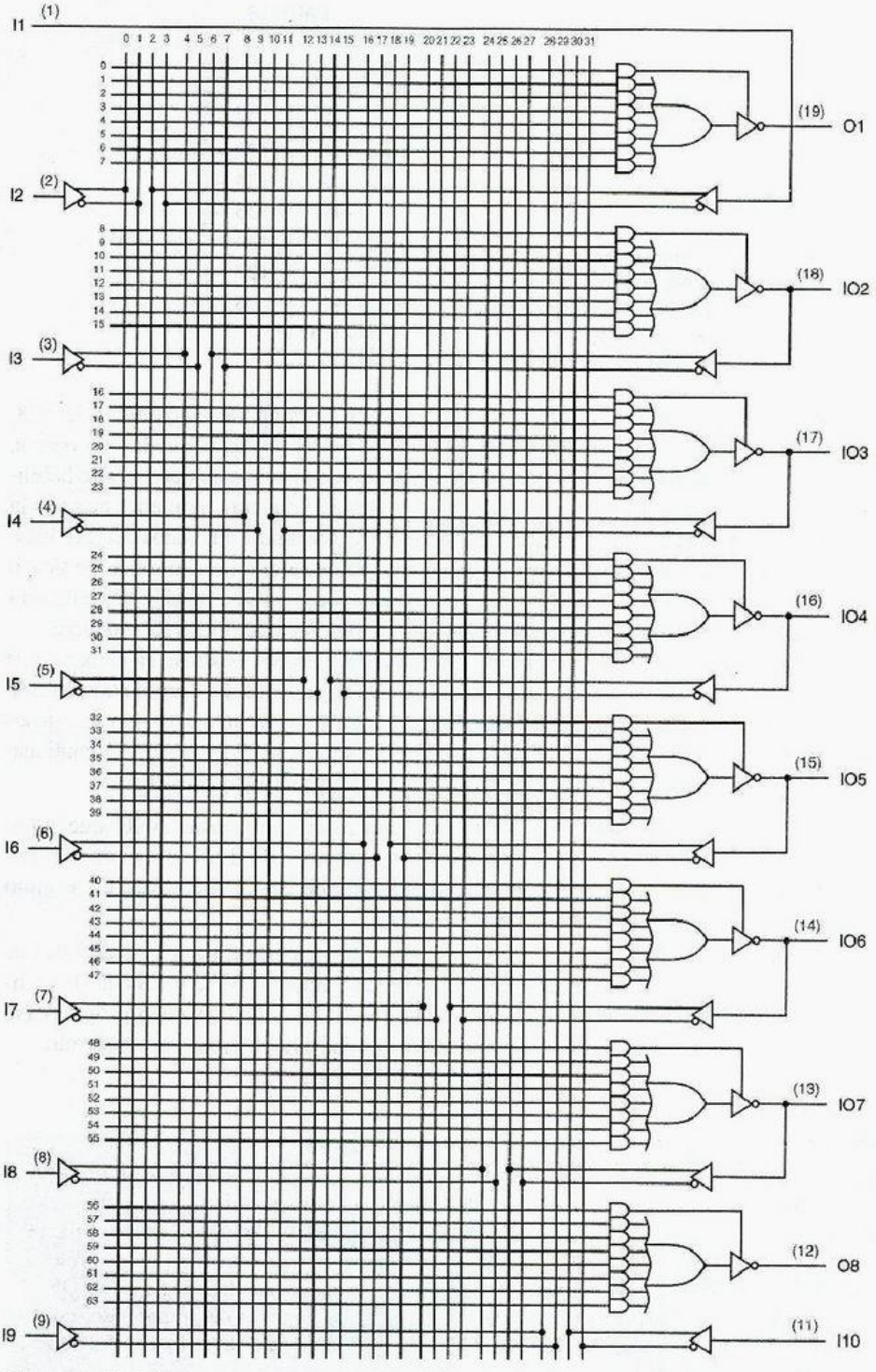


Figura 5-25 Diagrama lógico del PAL16L8.

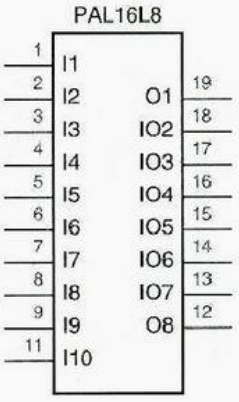


Figura 5-26
Símbolo lógico tradicional para el PAL16L8.

compuerta de habilitación de salida

Ocho compuertas AND están asociadas con cada terminal de salida del PAL16L8. Siete de ellas proporcionan entradas para una compuerta OR fija de 7 entradas. La octava, que llamamos *compuerta de habilitación de salida*, está conectada a la entrada de habilitación de tres estados del buffer de salida; el buffer se habilita solamente cuando la compuerta de habilitación de salida tiene una salida 1. De este modo, una salida del PAL16L8 puede efectuar solamente las funciones lógicas que se pueden escribir como sumas de siete o menos términos de producto. Cada término de producto puede ser una función de cualquiera o todas las 16 entradas, pero solamente están disponibles siete términos de producto.

Aunque el PAL16L8 tiene hasta 16 entradas y 8 salidas, se construye en el interior de un encapsulado DIP, que solamente tiene 20 terminales, incluyendo dos para la alimentación y tierra (las terminales de las esquinas, 10 y 20). Esta magia es el resultado de seis terminales bidireccionales (13-18) que pueden emplearse como entradas, salidas o ambas. A continuación se resumen ésta y otras diferencias entre el PAL16L8 y una estructura PLA:

- El PAL16L8 tiene un arreglo OR fijo, con siete compuertas AND que están conectadas en forma permanente a cada compuerta OR. Las salidas de la compuerta AND no pueden compartirse; si dos compuertas OR necesitan un término de producto, éste debe generarse dos veces.
- Cada salida del PAL16L8 tiene una señal individual de habilitación de salida de tres estados, que a su vez está controlada por una compuerta AND dedicada (la compuerta de habilitación de salida). De esta forma, las salidas pueden programarse como siempre habilitadas, siempre deshabilitadas o habilitadas por un término de producto que involucra a los dispositivos de entrada.

¿SON ÚTILES LOS SIETE TÉRMINOS DE PRODUCTO?

El peor caso de función para el diseño AND-OR de dos niveles es una función (paridad) XOR de n entradas, la cual requiere 2^{n-1} términos de producto. Sin embargo, funciones menos perversas con más de siete términos de producto de un PAL16L8 pueden construirse con frecuencia mediante su descomposición en una estructura de 4 niveles (AND-OR-AND-OR) que puede realizarse con dos pasos por medio del arreglo AND-OR. Desgraciadamente, además de utilizar salidas PLD para los términos de primer paso, esto duplica el retardo, puesto que una entrada de primer paso debe pasar a través del PLD dos veces para propagarse hacia la salida.

¡COMBINACIONAL, NO COMBINATORIO!

Un paso *hacia atrás* en la introducción de MMI de los dispositivos PAL fue la popularización de la palabra "combinatorio" para describir circuitos combinacionales. Los circuitos *combinacionales* no tienen memoria; sus salidas en cualquier momento dependen de la combinación de entrada presente. A los ingenieros en computación bien entrenados, la palabra "combinatorio" debería evocarles vívidas imágenes de coeficientes binomiales, complejidad de resolución de problemas y del gran científico computacional Donald Knuth.

- Existe un inversor entre la salida de cada compuerta OR y la terminal externa del dispositivo.
- También se pueden utilizar como entradas seis de las terminales de salida, denominadas *terminales de E/S*. Esto proporciona muchas posibilidades para utilizar cada terminal de E/S, dependiendo de cómo se encuentre programado el dispositivo:

terminal de E/S

  - Si una compuerta de control de salida de la terminal de E/S produce un 0 constante, entonces se habilitará la salida y la terminal se empleará estrictamente como una entrada.
  - Si la señal de entrada en una terminal de E/S no es utilizada por ninguna compuerta del arreglo AND, entonces la terminal se puede utilizar estrictamente como una salida. Dependiendo de la programación de la compuerta de habilitación de salida, la salida siempre puede estar habilitada, o puede estar habilitada solamente para ciertas condiciones de entrada.
  - Si una compuerta de control de salida de la terminal de E/S produce un 1 constante, entonces la salida siempre estará habilitada, pero la terminal todavía se puede utilizar como una entrada. De este modo, las salidas se pueden utilizar para generar "términos de ayuda" de primer paso para las funciones lógicas que no se pueden realizar en un solo paso (con el número limitado de términos AND que están disponibles para una sola salida). Mostraremos un ejemplo de este caso en la página 365.
  - En otro caso, cuando una terminal de E/S siempre está habilitada como salida, la salida se puede utilizar como una entrada para las compuertas AND que afectan a la misma salida. Es decir, podemos incrustar un circuito secuencial de retroalimentación en un PAL16L8. Volveremos a analizar este caso en la sección 8.2.6.

El *PAL20L8* es otro PLD combinacional similar al PAL16L8, excepto que su encapsulado tiene cuatro terminales adicionales "sólo de entrada", y cada una de sus compuertas AND tiene ocho entradas adicionales. Su estructura de salida es la misma que la del PAL16L8.

PAL20L8

5.3.3 Dispositivos lógicos de arreglo genérico

En la sección 8.3 presentaremos los PLD secuenciales, que son dispositivos de lógica programable que proporcionan flip-flops para algunas o todas las salidas de compuerta OR. Estos dispositivos se pueden programar para que realicen una amplia variedad de funciones útiles de circuito secuencial.

VELOCIDAD DEL PLD COMBINACIONAL

La velocidad del PLD combinacional por lo regular se establece como un solo número que indique el retardo de propagación t_{PD}, desde cualquier entrada hacia cualquier salida, para cualquier dirección de transición. Los PLD se encuentran disponibles en una amplia gama de velocidades; las partes que se utilizan más a menudo funcionan a 10 ns. En 1998, los PLD combinacionales más rápidos disponibles incluían un PAL16L8 bipolar a 5 ns y un GAL22LV10 CMOS de 3.3 V a 3.5 ns.

lógica de arreglo genérico
dispositivo GAL
GAL16V8

Un tipo de PLD secuencial, que fue presentado por vez primera por la empresa Lattice Semiconductor, se conoce como *lógica de arreglo genérico* o *dispositivo GAL*, y es particularmente popular. Un tipo de dispositivo GAL simple, el *GAL16V8*, se puede configurar (mediante programación y un correspondiente patrón de fusibles) para simular un AND-OR, flip-flop y la estructura de salida de una amplia variedad de dispositivos PAL combinacionales y secuenciales, incluyendo el PAL16L8 que presentamos anteriormente. Además, el dispositivo GAL se puede borrar (en forma eléctrica) y volver a programar.

La figura 5-27 muestra el diagrama lógico para un GAL16V8, el cual se ha configurado como un dispositivo estrictamente combinacional, semejante al PAL16L8. Esta configuración se consigue mediante la programación de dos fusibles de "control de arquitectura", que no se muestran. En esta configuración, el dispositivo se conoce como un *GAL16V8C*.

GAL16V8C

El punto más importante que se advierte con relación al diagrama lógico del GAL16V8C, en comparación con el del PAL16L8 de la figura 5-25, es la inserción de una compuerta XOR entre cada salida OR y el buffer de salida de tres estados. Una entrada de la compuerta XOR se "eleva" a un valor 1 lógico pero es conectada a tierra (0) a través de un fusible. Si el fusible está intacto, la compuerta XOR deja pasar la salida de la compuerta OR sin modificar, pero si el fusible se funde, la compuerta XOR invierte la salida de la compuerta OR. Entonces, se puede decir que este fusible controla la *polaridad de salida* de la terminal correspondiente de salida.

polaridad de salida

El control de la polaridad de salida es una característica muy importante de los PLD modernos, incluyendo el GAL16V8. Como lo discutimos en la sección 4.6.2, dada una función lógica para minimizar, un compilador ABEL encuentra las expresiones mínimas de suma de productos tanto para la función como para su complemento. Si el complemento produce menos términos de producto, se puede utilizar si el fusible de polaridad de salida del GAL16V8 se configura para inversión. A menos que se cancele, el compilador realiza la mejor selección en forma automática y establece los patrones de fusibles apropiadamente.

PALCE16V8
GAL20V8
PALCE20V8

Diversas compañías fabrican un dispositivo que es equivalente al GAL16V8, denominada *PALCE16V8*. También existe un dispositivo GAL de 24 terminales, el *GAL20V8* o el *PALCE20V8*, que se pueden configurar para simular la estructura del PAL20L8 o cualquier PLD secuencial, como se describe en la sección 8.3.2.

NOTA LEGAL GAL es una marca registrada de Lattice Semiconductor, Hillsboro, OR 97124.

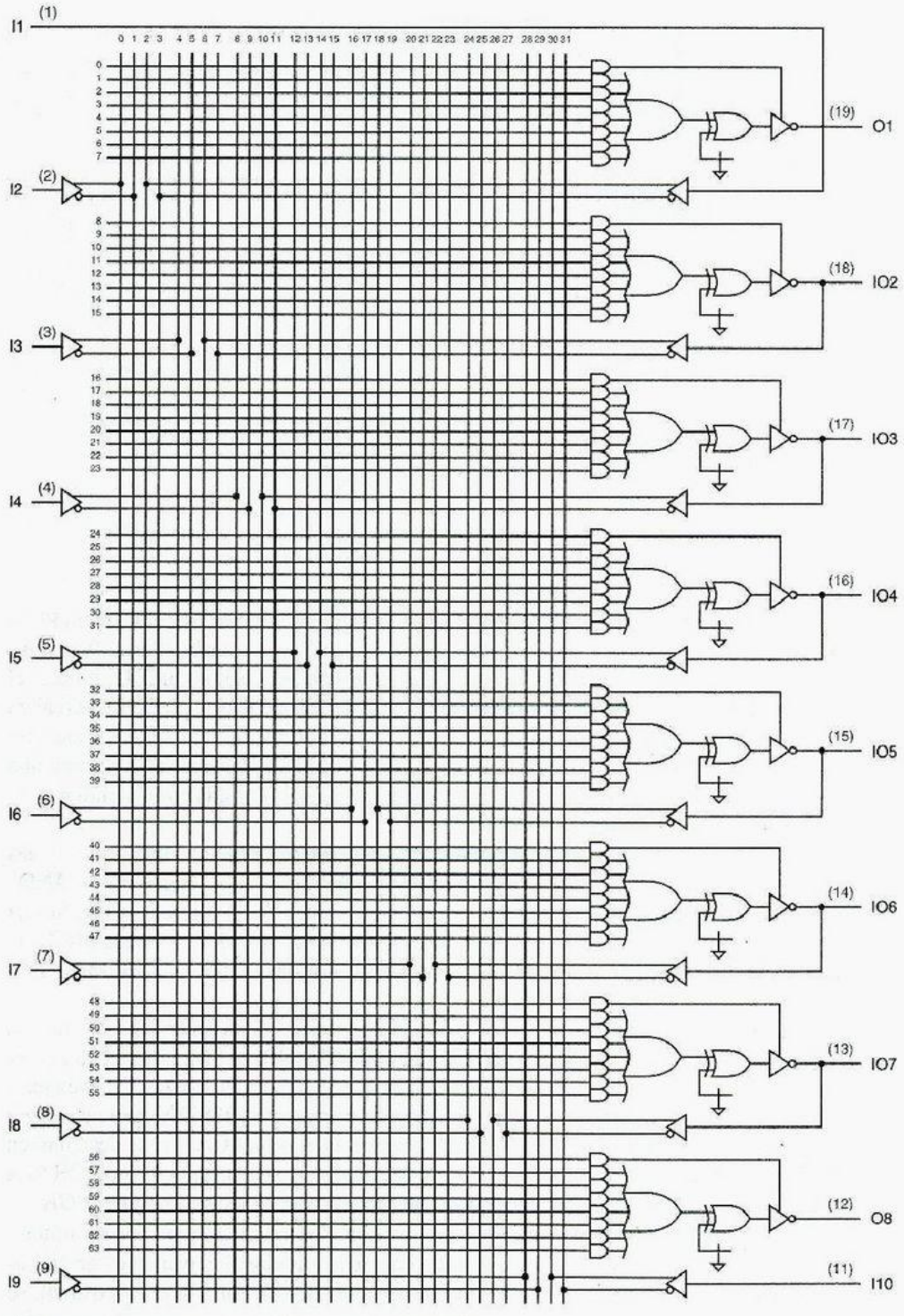


Figura 5-27 Diagrama lógico del GAL16V8C.

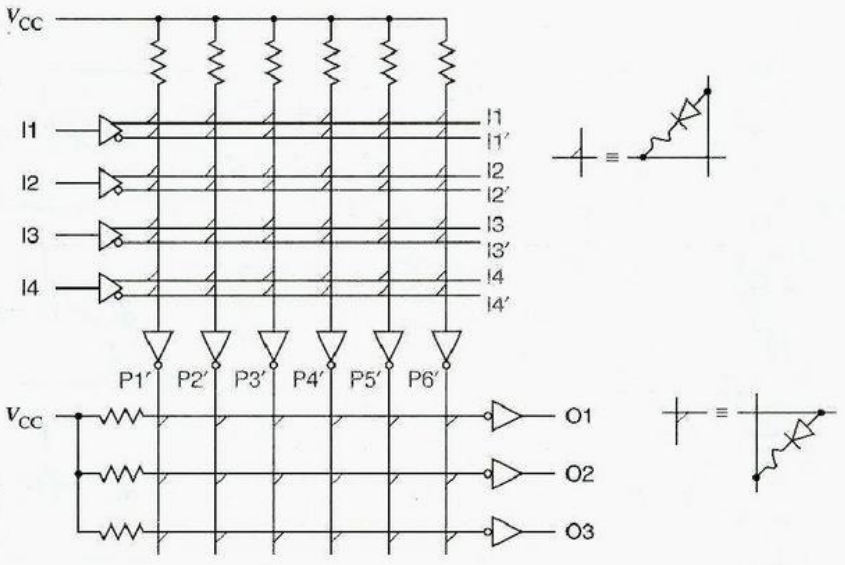


Figura 5-28
Un PLA de 4 X 3 construido utilizando compuertas de colector abierto tipo TTL y lógica de diodo.

*5.3.4 Circuitos PLD bipolares

Existen diversas tecnologías de circuito para construir y programar físicamente un PLD. Los primeros dispositivos PLA y PAL comerciales usaban circuitos bipolares. Por ejemplo, la figura 5-28 muestra cómo se puede construir con una tecnología TTL bipolar el circuito PLA 4 X 3 del ejemplo de la sección anterior. Cada conexión potencial se realiza con un diodo que está conectado en serie con un eslabón fusible, que puede estar presente o ausente. Si el eslabón está presente, entonces el diodo conecta su entrada en una función AND de diodo. Si el enlace está ausente, entonces la entrada correspondiente no tiene efecto sobre esa función AND.

Una función AND de diodo se realiza ahí, porque todas y cada una de las líneas horizontales de "entrada" que se conectan a través de un diodo a una línea vertical "AND" (en particular), deben estar en ALTO a fin de que la línea AND esté en ALTO. Si una línea de entrada está en BAJO, entonces lleva a nivel BAJO a todas las líneas AND con las que está conectada. Esta primera matriz de elementos de circuito que realizan la función AND se denomina el *plano AND*.

plano AND

Cada línea AND está seguida por un buffer inversor, de modo que se obtiene una función NAND global. Las salidas de las funciones NAND de primer nivel son combinadas por otro conjunto de funciones programables AND de diodo, que una vez más, están seguidas por inversores. El resultado es una estructura NAND-NAND de dos niveles que es funcionalmente equivalente a la estructura PLA AND-OR que se describió en la sección anterior. La matriz de los elementos de circuito que efectúan la función OR (o la segunda función NAND, dependiendo de cómo se vea) se conoce como *plano OR*.

plano OR

Un chip PLD bipolar se fabrica con todos sus diodos presentes, pero con un diminuto eslabón *fusible* que está en serie con cada uno de los diodos (los pequeños garabatos en la figura 5-28). Cuando se aplican patrones especiales de entrada al dispositivo, se

eslabón fusible

* En esta obra las secciones opcionales están marcadas con un asterisco.

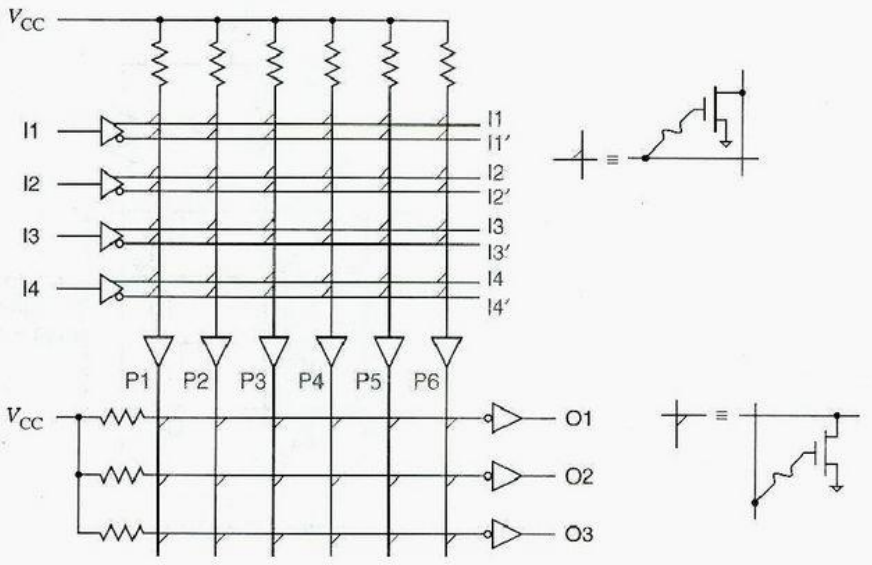


Figura 5-29
Un PLA de 4 × 3 construido utilizando lógica CMOS.

pueden seleccionar eslabones individuales mediante la aplicación de un voltaje alto (10-30 V), lo que vaporiza los eslabones seleccionados.

Los primeros PLD bipolares tenían problemas de confiabilidad. En ocasiones los patrones almacenados se modificaban debido a eslabones que se vaporizaban en forma incompleta, y por tanto podían "regenerarse". A veces ocurrían fallas intermitentes debido a los fragmentos que flotaban en el interior del encapsulado del CI. Sin embargo, estos problemas se han solucionado en gran parte, en la actualidad se utiliza una tecnología bastante confiable de eslabones fusibles en los PLD bipolares.

*5.3.5 Circuitos PLD CMOS

Aunque todavía se encuentran disponibles, los PLD bipolares se han reemplazado en gran medida por los PLD de CMOS con gran número de ventajas, incluyendo la facilidad de reprogramación y el reducido consumo de energía. La figura 5-29 muestra un diseño CMOS para el circuito PLA 4 × 3 de la sección 5.3.1.

En lugar de un diodo, se tiene un transistor de canal-*n* con una conexión programable en cada intersección entre una línea de entrada y una línea de palabra. Si la entrada es BAJO, entonces el transistor está "apagado", pero si la entrada se encuentra en ALTO, el transistor se "enciende", lo que lleva a la línea AND a BAJO. En términos generales, se obtiene una función AND de entrada invertida (es decir, una función NOR). Esto es semejante en estructura y función a una compuerta normal NOR CMOS de *k* entradas, excepto que se han reemplazado los *k* transistores de arranque de canal-*p* (que se conectan en serie) con un resistor de arranque pasivo. En la práctica, el arranque está formado por un transistor sencillo de canal-*p* con polarización constante.

Como se muestra a color gris en la figura 5-29, los efectos que se obtienen al utilizar una compuerta AND con entrada *inversora* se cancelan cuando se utilizan las líneas opuestas de entrada (complementadas) para cada entrada, en comparación con la

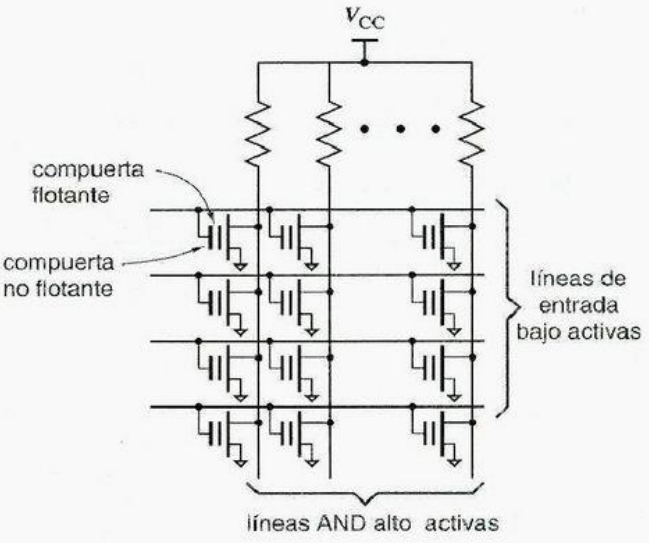


Figura 5-30
Plano AND de un EPLD utilizando transistores MOS de compuerta flotante.

figura 5-28. Cabe indicar que la conexión entre el plano AND y el plano OR no está invertida, de modo que el plano AND realiza una verdadera función AND.

Las salidas de las funciones AND de primer nivel se combinan en el plano OR por otro conjunto de funciones NOR con conexiones programables. La salida de cada función NOR está seguida de un inversor, de modo que se realiza una verdadera función OR, y la totalidad del PLA efectúa una función AND-OR.

En las tecnologías PLD CMOS, los eslabones programables que se muestran en la figura 5-29 no son del tipo fusible. En dispositivos no programables, tales como los chips personalizados VLSI, la presencia o ausencia de cada eslabón se establece en la mascarilla o plantilla de metal que se genera durante la fabricación del dispositivo. Sin embargo, la tecnología de programación que se utiliza con mayor frecuencia es la que se aplica en los EPLD CMOS, como se analiza a continuación.

dispositivo de lógica programable y borrable (EPLD)

Un *dispositivo de lógica programable borrable (EPLD, erasable programmable logic device)* se puede programar con cualquier configuración de enlace deseada, pero también se puede "borrar" para regresarlo a su estado original, ya sea de manera electrónica o exponiéndolo a luz ultravioleta. ¡No, el borrado no provoca que aparezcan o desaparezcan súbitamente los enlaces! En vez de ello, los EPLD utilizan una tecnología diferente, denominada "MOS de compuerta flotante".

transistor MOS de compuerta flotante

Como se muestra en la figura 5-30, un EPLD utiliza *transistores MOS de compuerta flotante*. Un transistor de esta naturaleza tiene dos compuertas. La compuerta "flotante" no se se conecta y está rodeada por un material aislante que tiene una impedancia extremadamente elevada. En el estado original, de fabricación, la compuerta flotante no tiene ninguna carga y no afecta la operación del circuito. En este estado, todos los transistores están "conectados"; es decir, existe un enlace lógico presente en todo punto de cruce en los planos AND y OR.

Para programar un EPLD, el programador aplica un voltaje elevado a la compuerta no flotante, en cada sitio donde no se desea un enlace lógico. Esto provoca una ruptura

temporal en el material aislante y permite que se acumule una carga negativa en la compuerta flotante. Cuando se elimina el voltaje alto, la carga negativa permanece en la compuerta flotante. Durante las operaciones siguientes, la carga negativa impide que el transistor se "encienda" cuando se aplica una señal de nivel ALTO a la compuerta no flotante; el transistor está desconectado del circuito.

Los fabricantes de EPLD afirman que un bit que esté programado en la forma apropiada conservará el 70% de su carga durante al menos 10 años, incluso si la parte se almacena a 125°C, de modo que para la mayor parte de las aplicaciones se puede considerar que la programación es permanente. Sin embargo, los EPLD también se pueden borrar.

Aun cuando algunos EPLD antiguos se ensamblaban en un encapsulado que tenía una tapa transparente y empleaban luz para borrarlos, en la actualidad los dispositivos más populares son *PLD eléctricamente borrables*. Las compuertas flotantes en un PLD eléctricamente borrable están rodeadas por una capa extremadamente gruesa de material aislante, y se pueden borrar mediante la aplicación de un voltaje que tiene la polaridad opuesta al voltaje de carga de la compuerta no flotante. De esta forma, el mismo equipo que se utiliza para programar un PLD también se puede utilizar para borrar un EPLD antes de programarlo.

PLD eléctricamente borrable

Los PLD "complejos", de mayor escala (CPLD), también utilizan la tecnología de programación de compuerta flotante. Incluso los dispositivos más grandes, frecuentemente conocidos como *arreglos de compuertas programables (FPGA, field-programmable gate arrays)*, utilizan celdas de memoria de lectura/escritura para controlar el estado de cada conexión. Las celdas de memoria de lectura/escritura son volátiles: no mantienen su estado cuando se desconecta la energía de alimentación. Por consiguiente, la primera vez que se aplica la energía de alimentación al FPGA, se debe inicializar toda su memoria de lectura/escritura para un estado que especifica una memoria externa no volátil. Esta memoria por lo regular es ya sea un chip de memoria programable de sólo lectura (PROM, *programmable read-only memory*) que está unido directamente al FPGA, o forma parte de un subsistema microprocesador que inicializa al FPGA como parte de la inicialización total del sistema.

arreglo de compuerta programable (FPGA)

*5.3.6 Prueba y programación del dispositivo

Se utiliza un equipo especial para vaporizar fusibles, cargar transistores de compuerta flotante o hacer cualquier cosa que se necesite para programar un PLD. Este equipo, que se encuentra en nuestros días en casi todos los laboratorios de diseño digital e instalaciones de producción, se conoce como *programador de PLD* o un *programador de PROM*.

programador de PLD
programador de PROM

CAMBIANDO EL HARDWARE "SOBRE LA MARCHA"

Los PROM se utilizan normalmente para suministrar el patrón de conexión para un FPGA basado en memoria de lectura/escritura, pero también existen aplicaciones donde el patrón es en realidad leído de un disco flexible. ¿Usted sólo recibe un disquete con una nueva versión de software? Adivine qué, ¡usted obtiene también una nueva versión de hardware!

Este concepto nos conduce a la intrigante idea, que ya se utiliza en algunas aplicaciones, del "hardware reconfigurable", en que se redefine un subsistema de hardware, sobre la marcha, para optimizar su desempeño en la tarea que se realiza.

(Se puede utilizar con memorias programables de sólo lectura, "PROM", así como también para PLD.) Un equipo típico para la programación de PLDs incluye un conector o conectores donde se insertan las terminales de los dispositivos que se van a programar, y un programa que "descarga" los patrones de programación deseados en el equipo, por lo regular mediante la conexión del mismo a una PC.

Los equipos programadores para PLD hacen que el PLD opere en un modo especial para realizar la programación. Por ejemplo, un equipo programador de PLD programa ocho fusibles de los PLD que se describen en este capítulo como se explica a continuación:

1. El voltaje de una terminal se eleva a un valor alto y predeterminado (del orden de 14 V) para llevar al dispositivo en el modo de programación.
2. Se selecciona un grupo de ocho fusibles al aplicar una "dirección" binaria a ciertas entradas del dispositivo. (Por ejemplo, el 82S100 tiene 1920 fusibles, y por tanto requeriría 8 entradas para seleccionar uno de 240 grupos de 8 fusibles).
3. Se aplica un valor de 8 bits a las *salidas* del dispositivo para especificar la programación deseada para cada fusible (las salidas se utilizan como entradas en el modo de programación).
4. Se eleva el voltaje de una segunda terminal a un valor alto y predeterminado durante una cantidad específica de tiempo (por decir 100 microsegundos) para programar los ocho fusibles.
5. Se baja el voltaje de la segunda terminal predeterminada a un nivel bajo (cercano a 0 V) para leer y verificar la programación de los ocho fusibles.
6. Se repiten los pasos 1 a 5 para cada grupo de ocho fusibles.

programabilidad en sistema

Muchos PLD, especialmente los CPLD más grandes, incluyen la función de *programabilidad en el sistema*. Esto quiere decir que el dispositivo se puede programar cuando está instalado y soldado en el sistema. En este caso, los patrones de fusión se aplican en serie al dispositivo, para esto se utilizan cuatro señales y terminales adicionales, denominadas el *puerto JTAG*, el cual está definido por el estándar IEEE 1149.1. Estas señales están definidas de tal modo que se pueden conectar "en cadena" varios dispositivos en la misma tarjeta de circuito impreso. Esto permite la selección y programación de los mismos durante el proceso de fabricación de la tarjeta, con lo que se utiliza solamente un puerto JTAG y un conector especial. No se necesita una fuente de alimentación especial de voltaje elevado; cada dispositivo utiliza un circuito interno de carga para generar el voltaje necesario para la programación.

puerto JTAG

Como se habrá notado en el paso anterior (el paso número 5), los patrones de fusión se verifican conforme se programan en el dispositivo. Si falla la programación de un fusible al primer intento, se repite la operación; si vuelve a fallar la programación después de algunos intentos, entonces se descarta el dispositivo (con frecuencia con gran perjuicio y premeditación).

Aunque la verificación del patrón de fusión de un dispositivo prueba que sus fusibles se encuentren adecuadamente programados, no asegura que el dispositivo realizará la función lógica que está especificada en esos fusibles. Esto es verdadero porque el dispositivo puede tener defectos internos, tales como conexiones faltantes entre los fusibles y los elementos del arreglo AND-OR.

La única forma de probar todos los defectos consiste en poner el dispositivo en su modo normal de operación, aplicar un conjunto de entradas lógicas normales, y observar

las salidas. El diseñador puede especificar los patrones requeridos de entrada y salida, también conocidos como vectores de prueba, como lo mostramos en la sección 4.6.7, o éstos se pueden generar en forma automática con la ayuda de un programa especial de generación de vectores de prueba. Sin tener en cuenta cómo se generaron los vectores de prueba, la mayor parte de los equipos programadores de PLD tienen la capacidad de aplicar entradas de vectores de prueba a un PLD y verificar sus salidas comparándolas con los resultados esperados.

fusible de seguridad

La mayor parte de los PLD tienen un *fusible de seguridad* que, cuando se programa, deshabilita la capacidad de leer los patrones de fusión del dispositivo. Los fabricantes pueden programar este fusible para evitar que otros lean los patrones de fusión del PLD para copiar el diseño de su producto. Incluso si se programa el fusible de seguridad, los vectores de prueba siguen funcionando, de modo que el PLD se puede verificar.

5.4 Decodificadores

decodificador

Un *decodificador* es un circuito lógico con varias entradas y salidas que convierte las entradas codificadas en salidas codificadas, donde los códigos de entrada y de salida son diferentes. El código de entrada generalmente tiene menos bits que el código de salida, y hay un mapeo unívoco desde las palabras del código de entrada hacia las palabras del código de salida. En un *mapeo unívoco*, cada palabra de código de entrada produce una palabra de código de salida diferente.

mapeo unívoco

La estructura general de un circuito decodificador se ilustra en la figura 5-31. Las entradas de habilitación, si se hallan presentes, deben ser afirmadas para que el decodificador efectúe su función de mapeo normal. De otro modo, el decodificador mapea todas las palabras del código de entrada en una sola, "deshabilitada", palabra de código de salida.

El código de entrada que se utiliza con mayor frecuencia es un código binario de n bits, donde una palabra de n bits representa uno de 2^n diferentes valores codificados, normalmente los enteros de 0 hasta $2^n - 1$. En ocasiones un código binario de n bits es truncado para representar menos de 2^n valores. Por ejemplo, en el código BCD, las combinaciones de 4 bits 0000 hasta 1001 representan los dígitos decimales 0 a 9, y las combinaciones 1010 hasta 1111 no se utilizan.

El código de salida que se utiliza con mayor frecuencia es un *código 1 de m*, que contiene m bits, donde un bit está activo en todo momento. De este modo, en un código 1 de 4 con salidas activas altas, las palabras de código son 0001, 0010, 0100 y 1000. Con salidas activas bajas, las palabras del código son 1110, 1101, 1011 y 0111.

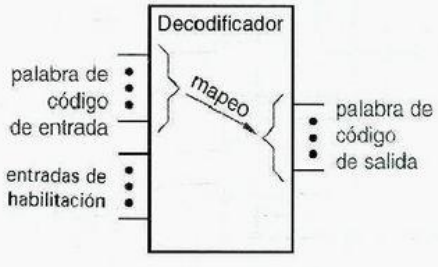


Figura 5-31
Estructura de circuito decodificador.

Tabla 5-4
Tabla de verdad para un decodificador binario de 2 a 4.

Entradas			*Salidas*			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

5.4.1 Decodificadores binarios

decodificador binario

El circuito decodificador más común es un decodificador de n a 2^n o *decodificador binario.* Un decodificador de esta clase tiene un código de entrada binario de n bits, y un código de salida 1 de 2^n. Se utiliza un decodificador binario cuando se necesita activa exactamente una de las 2^n salidas basado en un valor de entrada de n bits.

Por ejemplo, la figura 5-32(a) muestra las entradas y salidas, y la tabla 5-4 es la tabla de verdad de un decodificador de 2 a 4. La palabra de código de entrada I1, I0 representa un entero en el intervalo de 0 a 3. La palabra de código de salida Y3, Y2, Y1, Y0 tiene Yi igual a 1 si y sólo si la palabra de código de entrada es la representación binaria de i y la *entrada de habilitación* EN es 1. Si EN es 0, entonces todas las salidas son 0. Un circuito a nivel de compuerta para el decodificador 2 a 4 se muestra en la figura 5-32b). Cada compuerta AND *decodifica* una combinación de la palabra de código de entrada I1, I0.

entrada habilitada

decodificar

La tabla de verdad del decodificador binario introduce una notación "sin importancia" para combinaciones de entrada. Si uno o más valores de entrada no afectan los valores de salida para alguna combinación de las entradas restantes, se marcan con una "x" para esa combinación de entrada. Esta convención puede reducir en gran medida el número de renglones en la tabla de verdad, además de hacer las funciones de las entradas más claras.

El código de entrada de un decodificador binario de n bits no necesita representar los enteros desde 0 hasta $2^n - 1$. Por ejemplo, la tabla 5-5 muestra la salida en código

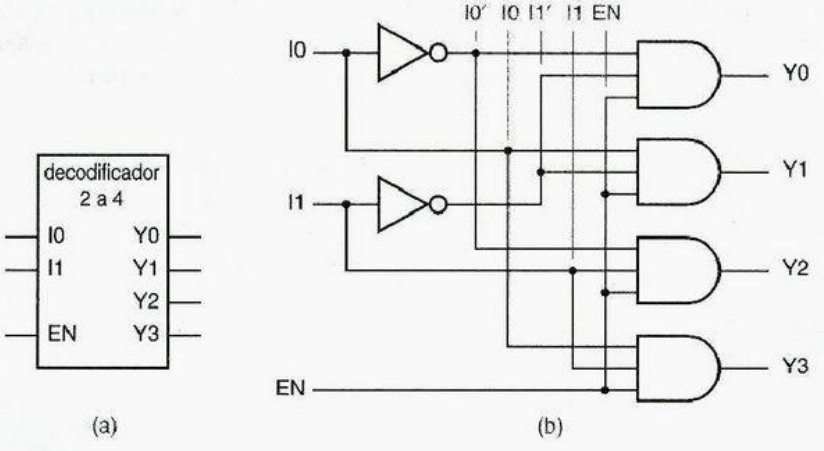


Figura 5-32
Un decodificador de 2 a 4: (a) entradas y salidas; (b) diagrama lógico.

Posición del disco	I2	I1	I0	*Salida decodificador binario*
0°	0	0	0	Y0
45°	0	0	1	Y1
90°	0	1	1	Y3
135°	0	1	0	Y2
180°	1	1	0	Y6
225°	1	1	1	Y7
270°	1	0	1	Y5
315°	1	0	0	Y4

Tabla 5-5
Codificación de posición para un disco de codificación mecánica de 3 bits.

Gray de 3 bits de un disco mecánico de codificación con ocho posiciones. Las ocho posiciones del disco se pueden decodificar con la ayuda de un decodificador binario de 3 bits, con la asignación apropiada de señales hacia las salidas del decodificador, como se ilustra en la figura 5-33.

Tampoco es necesario emplear todas las salidas de un decodificador, o incluso decodificar todas las combinaciones posibles de entrada. Por ejemplo, un *decodificador decimal o BCD* decodifica solamente las primeras diez combinaciones de entrada binaria 0000-1001 para producir las salidas Y0–Y9.

decodificador decimal
decodificador BCD

5.4.2 Símbolos lógicos para elementos de mayor escala

Antes de describir algunos decodificadores MSI de la serie 74 (que están disponibles en el mercado), necesitamos discutir los lineamientos generales para dibujar los símbolos lógicos de elementos lógicos de mayor escala.

La regla más básica indica que los símbolos lógicos se dibujen con las entradas a la izquierda y las salidas a la derecha. Los extremos superior e inferior de un símbolo lógico normalmente no se utilizan para realizar conexiones de señal. Sin embargo, en ocasiones, las conexiones explícitas de alimentación y tierra se muestran en la parte superior e inferior, especialmente si estas conexiones se realizan en las terminales "no estándar". (En la mayoría de las partes MSI, las terminales de alimentación y tierra se encuentran en las esquinas del encapsulado, por ejemplo, en las terminales 8 y 16 de un encapsulado DIP de 16 terminales.)

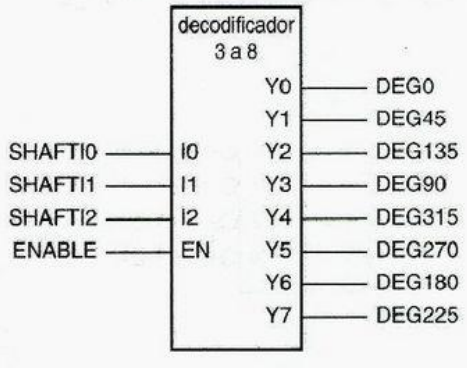


Figura 5-33
Uso de un decodificador binario 3 a 8 para decodificar un código Gray.

FAMILIAS LÓGICAS La mayor parte de las compuertas lógicas y elementos de escala más grande se encuentran disponibles en una amplia variedad de familias CMOS y TTL, muchas de las cuales describimos en las secciones 3.8 y 3.11. Por ejemplo, los 74LS139, 74S139, 74ALS139, 74AS139, 74F139, 74HC139, 74HCT139, 74ACT139, 74AC139, 74FCT139, 74AHC139, 74AHCT139, 74HLC139, 74LVC139 Y 74VHC139 son todos decodificadores de 2 a 4 con la misma función lógica, pero en familias CMOS y TTL eléctricamente diferentes y en ocasiones en diferentes encapsulados. Además, elementos lógicos "macro" con los mismos nombres de terminales y funciones que las del '139 y otros populares dispositivos de la serie 74 se encuentran disponibles como bloques de construcción en la mayoría de los ambientes de diseño FPGA y ASIC.

A lo largo de este texto, utilizaremos "74x" como un prefijo genérico. Y a veces omitiremos el prefijo y escribiremos; por ejemplo, '139. En un diagrama esquemático real para un circuito que se vaya a construir o simular, se debería incluir el número de parte completo, puesto que la temporización, carga y características de encapsulado dependen de la familia.

Como los símbolos de compuerta, los símbolos lógicos para elementos de mayor escala asocian un nivel activo con cada terminal. Con respecto a los niveles activos, es importante utilizar una convención consistente para nombrar las señales internas y las terminales externas.

Los elementos de mayor escala casi siempre tienen sus nombres de señal, los cuales están definidos en términos de funciones realizadas *dentro* de sus esquemas simbólicos, como se explica en la sección 5.1.4. Por ejemplo, la figura 5-34(a) muestra el símbolo lógico para una sección de un decodificador de 2 a 4 dual 74x139, una parte MSI que describiremos exhaustivamente en la siguiente subsección. Cuando la entrada G es asertiva, una de las salidas Y0-Y3 es asertiva, como se selecciona mediante un código de 2 bits aplicado a las entradas A y B. Es evidente por el símbolo, que la terminal de entrada G y todas las terminales de salida son activas bajas.

Cuando aparece el símbolo 74x139 en el diagrama lógico en una aplicación real, sus entradas y salidas tienen señales conectadas a otros dispositivos, y cada una de tales señales tienen un nombre que indica su función en la aplicación. Sin embargo, cuando describimos el 74x139 aisladamente, todavía podemos desear un nombre para la señal en cada terminal externa. La figura 5-34(b) muestra nuestra convención de nombrarlo en este caso. A las terminales activas altas se les da el mismo nombre de la señal interna, mientras que las terminales activas bajas tiene el nombre de la señal interna seguido por el sufijo "_L".

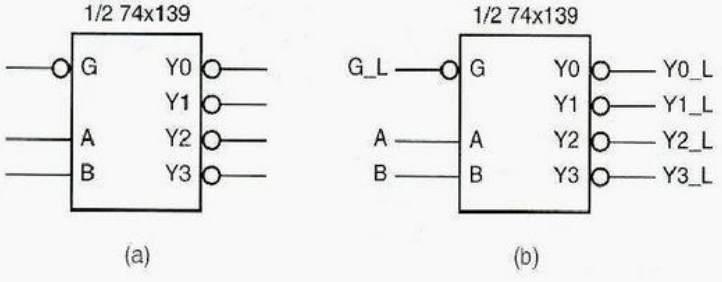


Figura 5-34
Símbolo lógico para una mitad de un decodificador de 2 a 4 dual 74x139: a) símbolo convencional; b) nombres de señal predeterminados asociados con terminales externas.

SÍMBOLOS LÓGICOS ESTÁNDARES DE IEEE A lo largo de este libro, empleamos símbolos "tradicionales" para elementos lógicos de mayor escala. El estándar IEEE utiliza símbolos algo diferentes para elementos lógicos de mayor escala. Los símbolos IEEE estándar, además de los pros y los contras de los símbolos tradicionales contra los de IEEE, se discuten en una guía disponible en el sitio web www.ddpp.com.

5.4.3 El decodificador de 2 a 4 dual 74x139

74x139

Dos decodificadores de 2 a 4 independientes e idénticos están contenidos en una sola parte MSI, el *74x139*. El diagrama de circuito a nivel de compuerta para este CI se muestra en la figura 5-35(a). Nótese que las salidas y la entrada de habilitación del '139 son activas bajas. La mayoría de los decodificadores MSI fueron originalmente diseñados con salidas activas bajas puesto que las compuertas inversoras TTL son por lo general más rápidas que las no inversoras. También advierta que el '139 tiene inversores extra en sus entradas de selección. Sin estos inversores, cada entrada de selección presentaría tres cargas de CA o CD en vez de una, consumiendo mucho más del fan out de salida del dispositivo que controla.

Figura 5-35 El decodificador de 2 a 4 dual 74x139: (a) diagrama lógico, incluyendo los números de terminales para un encapsulado en doble línea estándar de 16 terminales; (b) símbolo lógico tradicional; (c) símbolo lógico para un decodificador.

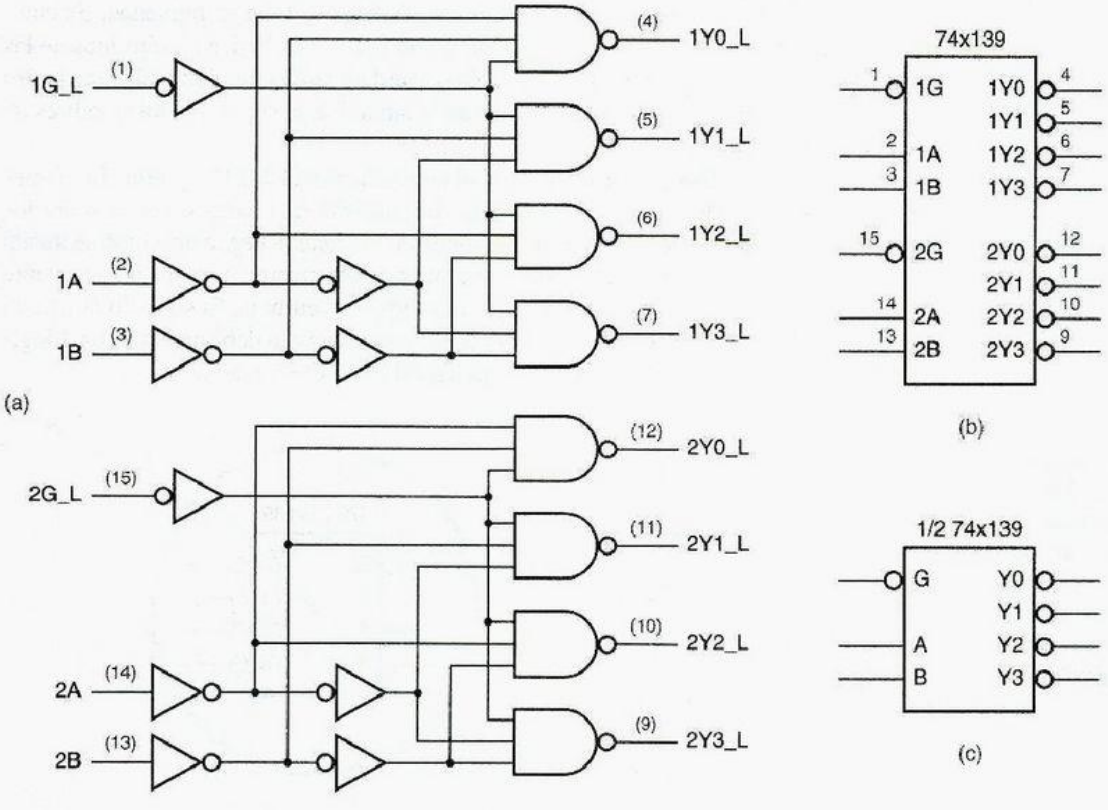


Tabla 5-6
Tabla de verdad para una mitad de un decodificador de 2 a 4 dual 74x139.

Entradas			*Salidas*			
G_L	B	A	Y3_L	Y2_L	Y1_L	Y0_L
1	x	x	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Un símbolo lógico para el 74x139 se ilustra en la figura 5-35(b). Nótese que todos los nombres de señal dentro del esquema del símbolo son activos altos (sin "_L") y que las burbujas de inversión indican entradas y salidas activas bajas. Con frecuencia un esquema puede emplear un símbolo genérico para apenas un decodificador, la mitad del '139, como se muestra en (c). En este caso, la asignación de la función genérica a una u otra mitad de un encapsulado '139 en particular puede ser diferida hasta que el esquema se complete.

La tabla 5-6 es la tabla de verdad para un decodificador tipo 74x139. Las tablas de verdad en los libros de datos de algunos fabricantes utilizan L y H para indicar los niveles de voltaje de señal de entrada y salida, de modo que no pueda haber ninguna ambigüedad acerca de la función eléctrica del dispositivo; una tabla de verdad escrita de esta manera se conoce a menudo como una *tabla de función*. Sin embargo, puesto que utilizamos la lógica positiva a lo largo de este libro, podemos emplear 0 y 1 sin ambigüedad. En cualquiera de los casos, la tabla de verdad proporciona la función lógica en términos de las *terminales externas* del dispositivo. Una tabla de verdad para la función realizada *dentro* del esquema del símbolo tendría el aspecto de la tabla 5-4, excepto que los nombres de señal de entrada serían G, B, A.

tabla de funciones

Algunos diseñadores lógicos dibujan el símbolo para el 74x139 y otras funciones lógicas sin burbujas de inversión. En vez de ello, utilizan una barra superior sobre los nombres de señal dentro del esquema del símbolo para indicar la negación, como se ilustra en la figura 5-36(a). Esta notación es consistente consigo misma, pero es inconsistente con nuestros estándares de dibujo para el diseño lógico en burbuja. El símbolo mostrado en (b) es absolutamente *incorrecto*: de acuerdo con este símbolo debe aplicarse un 1 lógico, no el 0, a la terminal de habilitación para habilitar el decodificador.

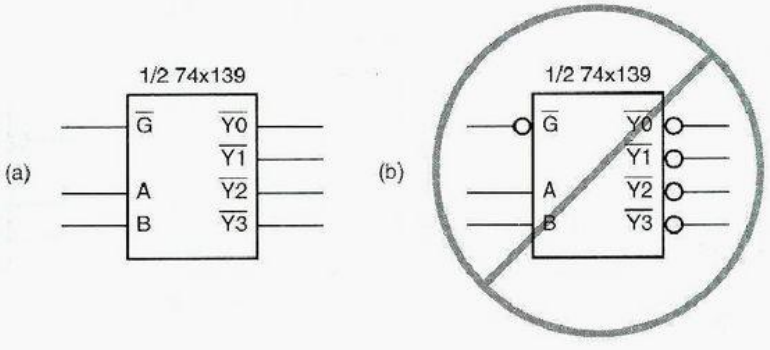


Figura 5-36
Otras maneras de simbolizar un 74x139: a) correcta pero no recomendable; b) incorrecta debido a las negaciones dobles.

NOMBRES MALOS

Algunas hojas de especificaciones de los fabricantes tienen inconsistencias semejantes a la figura 5-36(b). Por ejemplo, la hoja de datos de Texas Instruments para el 74AHC139 utiliza nombres activos bajos como 1G para las entradas de habilitación, con una barra superior para indicar una terminal activa baja, pero nombres activos altos como 1Y0 para todas las terminales de salida activas bajas. Por otro lado, la hoja de datos de Motorola para el 74VHC139 utiliza correctamente barras superiores sobre los nombres tanto para las entradas como para las salidas de habilitación, pero las barras superiores son escasamente visibles en la tabla de funciones del dispositivo, a consecuencia de un problema tipográfico.

He vivido la experiencia de construir una tarjeta de circuito impreso con muchas copias de un nuevo dispositivo, que adquirí con un distribuidor cuya documentación indicaba con claridad que una entrada en particular era activa baja, solamente para averiguar, hasta el primer encendido, que la entrada era activa alta.

La moraleja de esta historia es que usted tiene que estudiar la descripción de cada dispositivo para saber lo que en realidad está pasando. Y si trabaja con un dispositivo que pertenece a una marca nueva, ya sea de un distribuidor comercial o del grupo ASIC de su propia compañía, debería verificar dos veces todas las polaridades de señal y asignaciones de terminales antes de comprometerse a hacer un PCB. Sin embargo, esté seguro de que los nombres de las señales en este texto son consistentes y correctos.

5.4.4 El decodificador de 3 a 8 74x138

El 74x138 es un decodificador de 3 a 8 MSI disponible comercialmente cuyo símbolo y diagrama de circuito al nivel de compuerta se ilustran en la figura 5-37; su tabla de verdad se proporciona en la tala 5-7. Como el 74x139, el 74x138 tiene salidas activas bajas, y tiene tres entradas de habilitación (G1, /G2A, /G2B), todas ellas deben ser asertivas para que la salida seleccionada sea asertiva.

La función lógica del '138 es directa: una salida se afirma si y sólo si el decodificador está habilitado y la salida es seleccionada. De este modo, podemos fácilmente escribir ecuaciones lógicas para una señal de salida interna tal como Y5 en términos de las señales de entrada internas:

$$Y5 = \underbrace{G1 \cdot G2A \cdot G2B}_{\text{habilitado}} \cdot \underbrace{C \cdot B' \cdot A}_{\text{seleccionado}}$$

Sin embargo, debido a las burbujas de inversión, tenemos las siguientes relaciones entre señales internas y externas:

$$\begin{aligned} G2A &= G2A\_L' \\ G2B &= G2B\_L' \\ Y5 &= Y5\_L' \end{aligned}$$

Por lo tanto, si estamos interesados, podemos escribir la ecuación siguiente para la señal de salida externa Y5_L en términos de señales de entrada externas:

$$\begin{aligned} Y5\_L = Y5' &= (G1 \cdot G2A\_L' \cdot G2B\_L' \cdot C \cdot B' \cdot A)' \\ &= G1' + G2A\_L + G2B\_L + C' + B + A' \end{aligned}$$

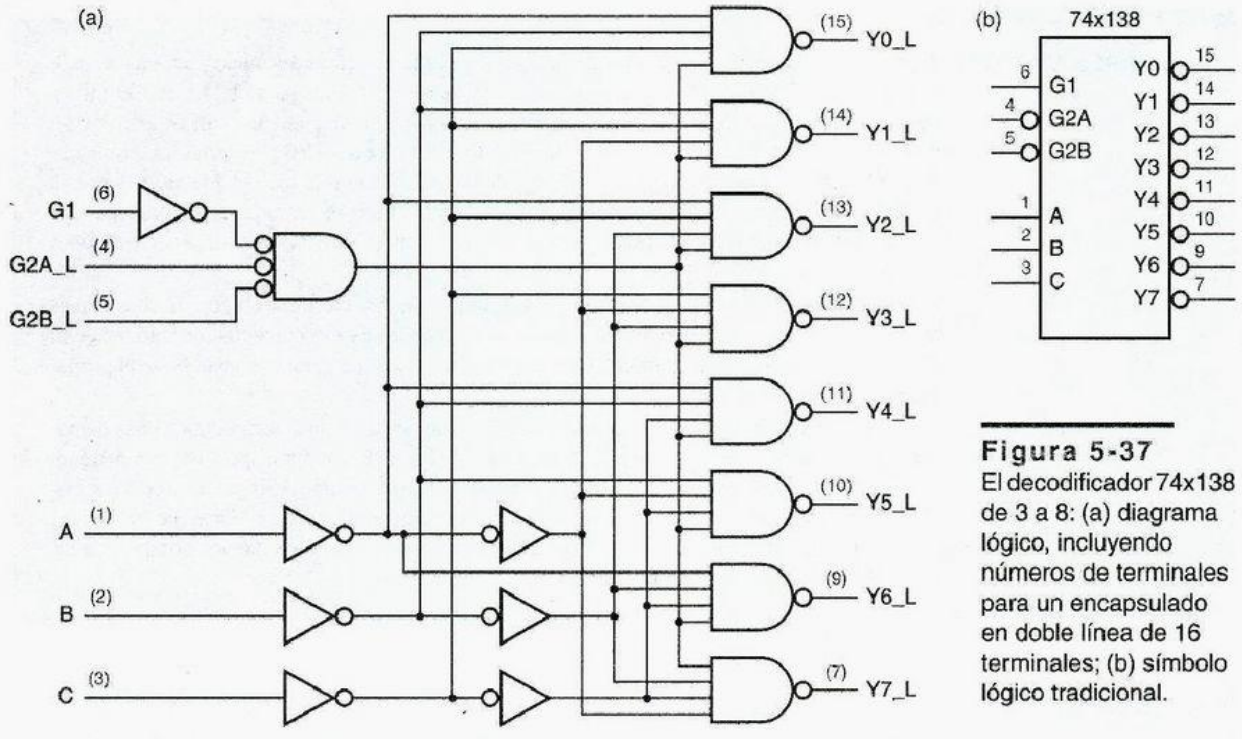


Figura 5-37
El decodificador 74x138 de 3 a 8: (a) diagrama lógico, incluyendo números de terminales para un encapsulado en doble línea de 16 terminales; (b) símbolo lógico tradicional.

Tabla 5-7 Tabla de verdad correspondiente a un decodificador 74x138 de 3 a 8.

Entradas						*Salidas*							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

En la superficie, esta ecuación no se parece a lo que podríamos esperar para un decodificador, puesto que es una suma lógica en vez de un producto. Sin embargo, si se practica el diseño lógico en burbuja, no tiene que preocuparse por esto; sólo proporcione un nombre activo bajo a la señal de salida y recuerde que es activa baja cuando la conecte a otras entradas.

5.4.5 Decodificadores binarios en cascada

Los decodificadores binarios múltiples pueden utilizarse para decodificar palabras de código más grandes. La figura 5-38 muestra cómo dos decodificadores de 3 a 8 pueden combinarse para hacer un decodificador de 4 a 16. La disponibilidad de entradas de habilitación tanto activas altas como activas bajas en el 74x138 hace posible habilitar una u otra directamente basándose en el estado del bit de entrada más significativo. El decodificador superior (U1) está habilitado cuando N3 o 0, mientras que el inferior (U2) se encuentra habilitado cuando N3 es 1.

Para manejar palabras de código incluso más grandes, los decodificadores binarios pueden conectarse en cascada de manera jerárquica. La figura 5-39 ilustra cómo emplear la mitad de un 74x139 para decodificar los dos bits de mayor orden de una palabra de código de 5 bits, habilitando así uno de cuatro 74x138 que decodifican los tres bits de orden menor.

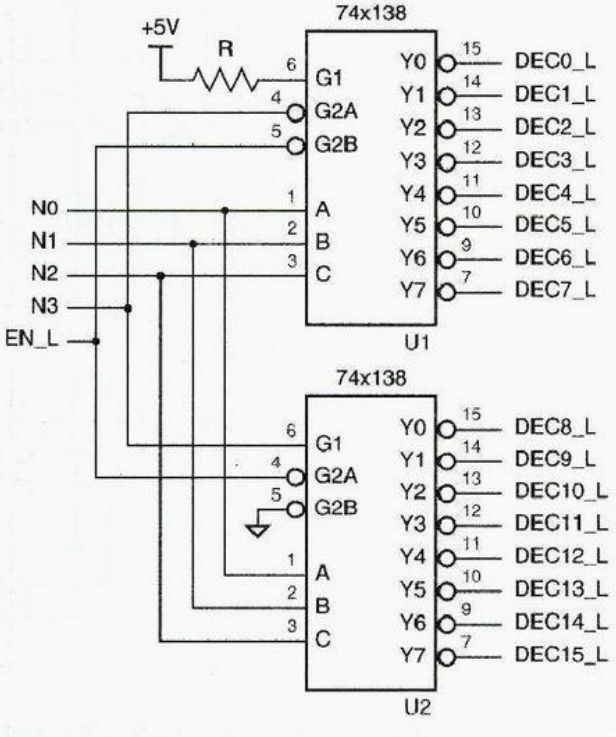


Figura 5-38
Diseño de un decodificador de 4 a 16 utilizando los 74x138.

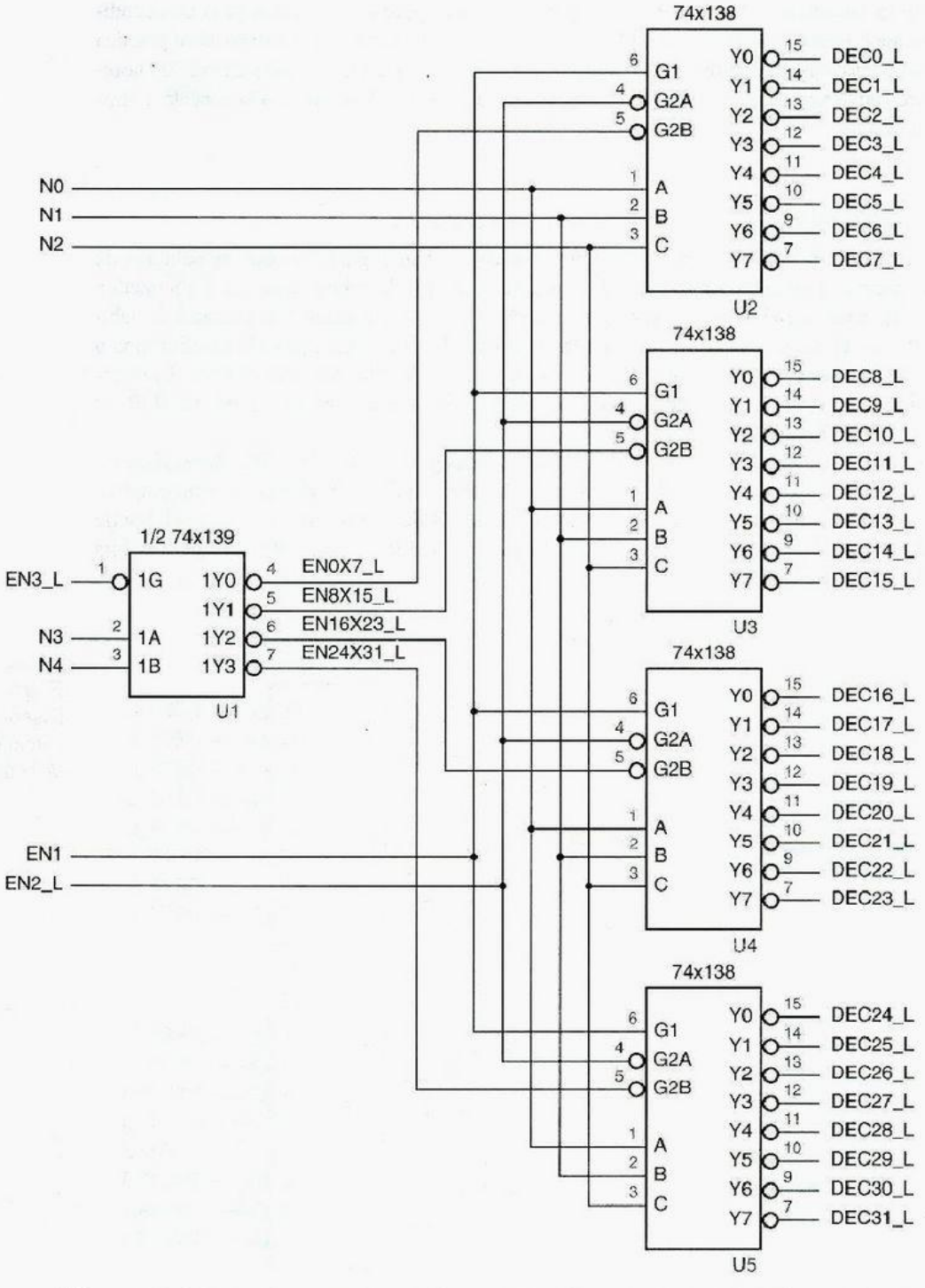


Figura 5-39 Diseño de un decodificador de 5 a 32 utilizando los 74x138 y un 74x139.

5.4.6 Decodificadores en ABEL y en los PLD

Nada en diseño lógico es más fácil que escribir las ecuaciones PLD para un decodificador. Puesto que la expresión lógica para cada salida es por lo regular sólo un término de producto simple, los decodificadores se adaptan con mucha facilidad a los PLD y utilizan pocos recursos de términos de producto.

Por ejemplo, la tabla 5-8 es un programa ABEL para un decodificador binario tipo 3 a 8 74x138 como se realiza en un PAL16L8. Advierta que algunas de las terminales de entrada y todas las terminales de salida tienen nombres activos bajos (sufijo "_L") en las declaraciones de terminales, correspondientes al diagrama lógico en la figura 5-37. Sin embargo, el programa define también un nombre activo alto correspondiente para cada señal de modo que las ecuaciones pueden escribirse todas "naturalmente", en términos de señales activas altas. Una forma alterna de conseguir el mismo efecto se describe en el cuadro de la página 363.

Tabla 5-8 Un programa ABEL para un 74x138 como decodificador binario de 3 a 8.

```
module Z74X138
title '74x138 Decoder PLD
J. Wakerly, Stanford University'
Z74X138 device 'P16L8';

" Input and output pins
A, B, C, G2A_L, G2B_L, G1                          pin 1, 2, 3, 4, 5, 6;
Y0_L, Y1_L, Y2_L, Y3_L, Y4_L, Y5_L, Y6_L, Y7_L     pin 19..12 istype 'com';

" Active-high signal names for readability
G2A = !G2A_L;
G2B = !G2B_L;
Y0 = !Y0_L;
Y1 = !Y1_L;
Y2 = !Y2_L;
Y3 = !Y3_L;
Y4 = !Y4_L;
Y5 = !Y5_L;
Y6 = !Y6_L;
Y7 = !Y7_L;

" Constant expression
ENB = G1 & G2A & G2B;

equations
Y0 = ENB & !C & !B & !A;
Y1 = ENB & !C & !B &  A;
Y2 = ENB & !C &  B & !A;
Y3 = ENB & !C &  B &  A;
Y4 = ENB &  C & !B & !A;
Y5 = ENB &  C & !B &  A;
Y6 = ENB &  C &  B & !A;
Y7 = ENB &  C &  B &  A;

end Z74X138
```

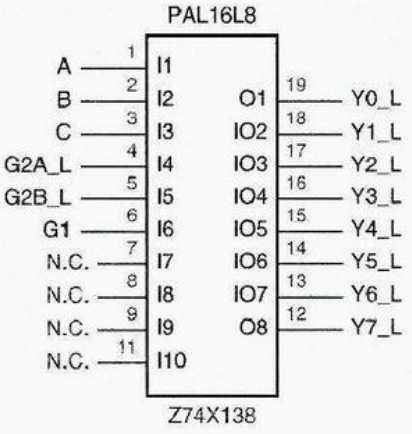


Figura 5-40
Diagrama lógico para el PAL16L8 que se utiliza como un decodificador 74x138.

También note que el programa ABEL define una expresión constante para ENB. Aquí, ENB no es una señal de entrada o salida, sino meramente un nombre definido por el usuario. En la sección equations, el compilador sustituye la expresión (G1 & G2A & G2B) dondequiera que aparezca ENB. La asignación de la expresión constante a un nombre definido por el usuario mejora la legibilidad y mantenimiento de este programa.

Si todo lo que se necesitaba era un '138, se estaría mejor empleando un '138 real en lugar de un PLD más costoso. Sin embargo, si usted necesita funcionalidad fuera del estándar, entonces el PLD puede por lo regular conseguirlo de manera mucho más económica y fácil que una solución basada en MSI/SSI. Por ejemplo, si necesitamos la funcionalidad de un '138 pero con salidas alto activas, necesitaremos solamente cambiar una línea en las declaraciones de terminal de la tabla 5-8:

```
Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7          pin 19..12 istype 'com';
```

(También, las definiciones originales de Y0–Y7 en la tabla 5-8 deben eliminarse.) Puesto que cada una de las ecuaciones requería un producto simple de seis variables (incluyendo las tres en la expresión ENB), cada ecuación complementada requiere una *suma* de seis términos de producto, menos que los siete disponibles en un PAL16L8. Si usamos un PAL16V8 u otro dispositivo con selección de polaridad de salida, entonces el compilador puede seleccionar emplear polaridad de salida no invertida solamente un término de producto por salida.

Tabla 5-9 Declaraciones alternas para un decodificador binario de 3 a 8 tipo 74x138.

```
" Input and output pins
A, B, C, !G2A, !G2B, G1                    pin 1, 2, 3, 4, 5, 6;
!Y0, !Y1, !Y2, !Y3, !Y4, !Y5, !Y6, !Y7     pin 19..12 istype 'com';

" Constant expression
ENB = G1 & G2A & G2B;
```

DEFINICIONES DE TERMINAL ACTIVA BAJA

ABEL nos permite utilizar un prefijo de inversión (!) en nombres de señal en las definiciones de terminales de un programa. Cuando se define un nombre de terminal con el prefijo de inversión, el compilador automáticamente antepone un prefijo de inversión al nombre de señal en cualquier parte que aparezca en otro lugar del programa. Si ya se encuentra invertido, se obtiene una doble inversión.

Esta característica se puede utilizar para definir una convención distinta pero consistente para la definición de entradas y salidas activas bajas; dando a cada señal activa baja un nombre activo alto, pero precediendo con el prefijo de inversión en su definición de terminal. Para el decodificador de 3 a 8 en la tabla 5-8, reemplazamos la primera parte del programa con el código mostrado en la tabla 5-9; la sección `equations` del programa permanece exactamente igual.

La elección de la convención que se debe emplear es una cuestión de gusto personal, aunque también depende de las capacidades de las herramientas CAD que se utilicen para dibujar los esquemas. Muchas herramientas nos permiten crear automáticamente un símbolo esquemático desde un bloque lógico que se define mediante un programa ABEL. Si la herramienta deja colocar burbujas de inversión sobre entradas y salidas seleccionadas del símbolo, entonces la convención en la tabla 5-9 produce un símbolo con nombres de señal activos altos dentro del contorno de función. Se pueden, entonces, especificar burbujas externas en señales activas bajas para obtener un símbolo que satisfaga las convenciones descritas en la sección 5.4.2 y mostradas en la figura 5-41(a).

Por otro lado, tal vez usted no pueda ni desee suministrar burbujas de inversión en símbolos creados en CAD. En ese caso, usted debería utilizar la convención en la tabla 5-8; esto produce un símbolo creado en CAD en el cual el nivel activo se indica por medio del nombre de la señal dentro del contorno de la función; no se necesitan burbujas de inversión externas. Esto se muestra en la figura 5-41(b). Observe que a diferencia de la figura 5-36(a) de la página 356, empleamos una convención basada en texto (`_L`) en vez de una barra superior sobre el nombre de la señal para indicar el nivel activo. Una convención basada en texto elegida apropiadamente proporciona transportabilidad entre diferentes herramientas de CAD:

Seleccionaremos algo arbitrariamente una convención a la otra en cada uno de los ejemplos de ABEL en el resto de este libro, únicamente para ayudarle a sentirse confortable con ambos enfoques.

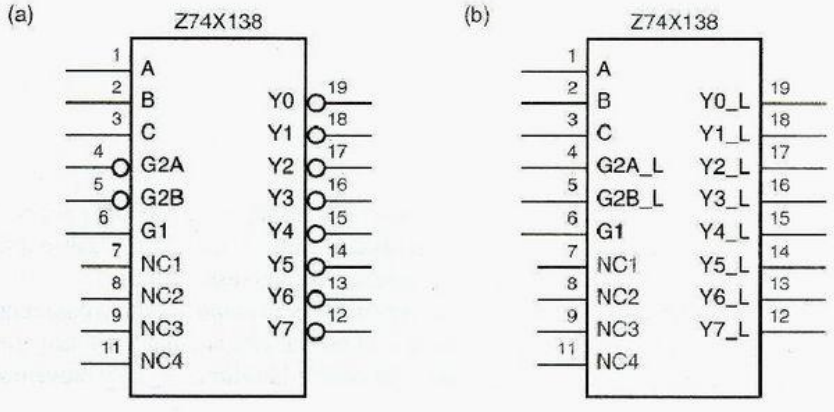


Figura 5-41
Posibles símbolos creados en CAD para el decodificador basado en PLD, tipo 74x138: a) basado en la tabla 5-9, después de la inserción manual de burbujas de inversión; b) basado en la tabla 5-8.

Otro cambio fácil es para proporcionar entradas habilitadas alternas que están relacionadas con OR con las entradas habilitadas principales. Para hacer esto, se necesita sólo definir terminales adicionales y modificar la definición de ENB:

```
EN1, EN2_L                          pin 7, 8;
...
EN2 = !EN2_L;
...
ENB = G1 & G2A & G2B # EN1 # EN2;
```

Este cambio expande el número de términos de producto por salida a tres, cada uno con una forma semejante a

```
Y0 = G1 & G2A & G2B & !C & !B &!A
   # EN1 & !C & !B & !A
   # EN2 & !C & !B & !A;
```

(Recuerde que el PAL16L8 tiene un inversor fijo y el PAL16V8 tiene un inversor seleccionable entre el arreglo AND-OR y la salida del PLD, de modo que la salida real es activa baja como se desea.)

Si se agregan las habilitadas extra a la versión del programa con salidas activas altas, entonces el PLD debe realizar el complemento de la expresión de productos de suma anterior. No es obvio de inmediato cuántos términos de producto tendrá esta expresión, y si se ajustará en un PAL16L8, pero podemos utilizar el compilador ABEL para obtener la respuesta para nosotros:

```
!Y0 = C # B # A # !G2B & !EN1 & !EN2
                # !G2A & !EN1 & !EN2
                # !G1  & !EN1 & !EN2;
```

La expresión tiene un total de seis términos de producto, de modo que se adapta en un PAL16L8.

Como un pellizco final, podemos agregar una entrada para controlar dinámicamente si la salida es activa alta o activa baja, y modificar todas las ecuaciones como sigue:

```
POL                 pin 9;
...
Y0 = POL $ (ENB & !C & !B & !A);
Y1 = POL $ (ENB & !C & !B &  A);
...
Y7 = POL $ (ENB &  C &  B &  A);
```

Como un resultado de la operación XOR, el número de términos de producto necesarios por salida se incrementa a 9, en cualquier polaridad de terminal de salida. De esta forma, incluso un PAL16V8 no puede implementar la función como está escrita.

salida auxiliar

La función todavía puede ser realizada si creamos una *salida auxiliar* para reducir la explosión de los términos de producto. Como se muestra en la tabla 5-10, asignamos una terminal de salida para la expresión ENB (perdiendo la salida Y7_L) y movemos la

Tabla 5-10 Fragmento de programa ABEL mostrando la lógica de dos pasos.

```
...
" Output pins
Y0_L, Y1_L, Y2_L, Y3_L          pin 19, 18, 17, 16 istype 'com';
Y4_L, Y5_L, Y6_L, ENB           pin 15, 14, 13, 12 istype 'com';

equations
ENB = G1 & G2A & G2B # EN1 # EN2;
Y0 = POL $ (ENB & !C & !B & !A);
...
```

ecuación ENB a la sección equations del programa. Esto reduce el requerimiento de término de producto a cinco en cualquier polaridad.

Aparte de sacrificar una terminal para la salida auxiliar, esta realización tiene la desventaja de ser más lenta. Cualquier cambio en las entradas a la expresión auxiliar debe propagarse a través del PLD dos veces, antes que alcance la salida final. Esto se conoce como la *lógica de dos pasos*. Muchas herramientas de síntesis PLD y FPGA pueden generar de manera automática lógica con dos o más pasos si una expresión requerida no puede realizarse sólo en un paso a través del arreglo lógico.

lógica de dos pasos

Los decodificadores se pueden personalizar de otras formas. Una personalización común es para que una salida simple decodifique más de una combinación de entrada. Por ejemplo, supongamos que se necesita generar un conjunto de señales habilitadas de acuerdo con la tabla 5-11. Puede aumentarse un decodificador 74x138 MSI como se muestra en la figura 5-42 para llevar a cabo la función requerida. Este enfoque, mientras que es potencialmente menos costoso que un PLD, tiene varias desventajas: requiere de retardo y componentes adicionales para crear las salidas requeridas, y no se puede modificar fácilmente.

CS_L	RD_L	A2	A1	A0	*Salida(s) para aserción*
1	x	x	x	x	ninguna
x	1	x	x	x	ninguna
0	0	0	0	0	BILL_L, MARY_L
0	0	0	0	1	MARY_L, KATE_L
0	0	0	1	0	JOAN_L
0	0	0	1	1	PAUL_L
0	0	1	0	0	ANNA_L
0	0	1	0	1	FRED_L
0	0	1	1	0	DAVE_L
0	0	1	1	1	KATE_L

Tabla 5-11 Tabla de verdad para una función de decodificador personalizada.

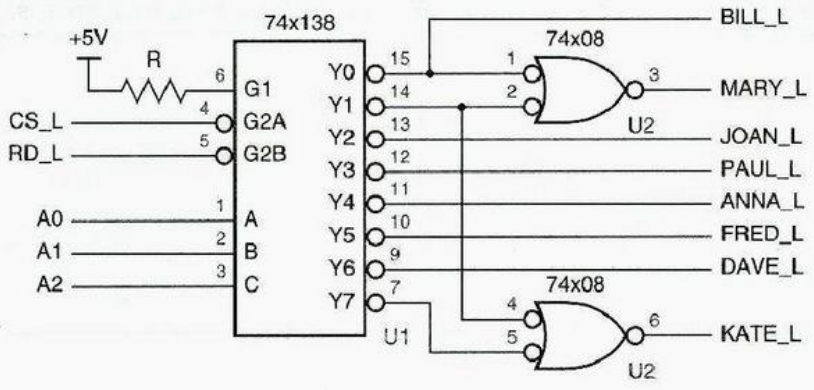


Figura 5-42
Circuito decodificador personalizado.

Una solución PLD al mismo problema se muestra en la tabla 5-12. Advierta que este programa utiliza la convención de nombrado de terminales de terminal activa baja descrita en el cuadro de la página 363 (usted debería sentirse cómodo con cualquier convención). Cada una de las últimas seis ecuaciones utiliza una compuerta AND simple en el PLD. El compilador ABEL también minimizará la ecuación MARY para emplear sólo una compuerta AND. Las señales de salida alto activas podrían obtenerse precisamente cambiando dos líneas en la sección de declaración:

```
BILL, MARY, JOAN, PAUL        pin 19, 18, 17, 16 istype 'com';
ANNA, FRED, DAVE, KATE        pin 15, 14, 13, 12 istype 'com';
```

Tabla 5-12 Ecuaciones ABEL para un decodificador personalizado.

```
module CUSTMDEC
title 'Customized Decoder PLD
J. Wakerly, Stanford University'
CUSTMDEC device 'P16L8';

" Input pins
!CS, !RD, A0, A1, A2          pin 1, 2, 3, 4, 5;
" Output pins
!BILL, !MARY, !JOAN, !PAUL    pin 19, 18, 17, 16 istype 'com';
!ANNA, !FRED, !DAVE, !KATE    pin 15, 14, 13, 12 istype 'com';

equations
BILL = CS & RD & (!A2 & !A1 & !A0);
MARY = CS & RD & (!A2 & !A1 & !A0 # !A2 & !A1 &  A0);
KATE = CS & RD & (!A2 & !A1 &  A0 #  A2 &  A1 &  A0);
JOAN = CS & RD & (!A2 &  A1 & !A0);
PAUL = CS & RD & (!A2 &  A1 &  A0);
ANNA = CS & RD & ( A2 & !A1 & !A0);
FRED = CS & RD & ( A2 & !A1 &  A0);
DAVE = CS & RD & ( A2 &  A1 & !A0);

end CUSTMDEC
```

Otra manera de escribir las ecuaciones se muestra en la tabla 5-13. En la mayoría de las aplicaciones, este estilo es más claro, especialmente si las entradas seleccionadas tienen significancia numérica.

Tabla 5-13 Ecuaciones ABEL equivalentes para un decodificador personalizado.

```
ADDR = [A2,A1,A0];

equations
BILL = CS & RD & (ADDR == 0);
MARY = CS & RD & (ADR == 0) # (ADDR == 1);
KATE = CS & RD & (ADR == 1) # (ADDR == 7);
JOAN = CS & RD & (ADDR == 2);
PAUL = CS & RD & (ADDR == 3);
ANNA = CS & RD & (ADDR == 4);
FRED = CS & RD & (ADDR == 5);
DAVE = CS & RD & (ADDR == 6);
```

5.4.7 Decodificadores en VHDL

Existen varias maneras de enfocar el diseño de decodificadores en VHDL. El enfoque más primitivo sería escribir un equivalente estructural de un circuito lógico decodificador, como la tabla 5-14 lo hace para el decodificador binario de 2 a 4 de la figura 5-32. Se supone que los componentes `and3` e `inv` ya existen en la tecnología destino. Naturalmente, esta conversión mecánica de un diseño existente en el equivalente de una lista de red elimina el propósito de emplear VHDL en primer lugar.

En su lugar, podríamos querer escribir un programa que utilice VHDL para hacer nuestro diseño de decodificador más comprensible y con mejor mantenimiento. La tabla 5-15 muestra un enfoque para escribir código para un decodificador binario de 3 a 8 equivalente al 74x138, que utiliza el estilo de flujo de datos de VHDL. Las entradas de dirección `A (2 downto 0)` y las salidas decodificadas activas bajas `Y_L (0 to 7)` son declaradas usando vectores para mejorar su legibilidad. Una instrucción `select` numera los ocho casos de decodificación y asigna el patrón apropiado de salida activa baja a una señal interna de 8 bits `Y_L_i`. Este valor se asigna a la salida del circuito real `Y_L` solamente si todas las entradas habilitadas son afirmadas.

Este diseño es un buen comienzo, y funciona, pero tiene una falla potencial. Los ajustes que determinan que dos entradas y todas las salidas son activas bajas están enterrados en la instrucción de asignación final. Aunque es cierto que la mayoría de los programas VHDL se escriben casi por completo con señales activas bajas, si definimos un dispositivo con terminales externas activas bajas, en realidad deberíamos manejarlas de una manera más sistemática y fácil de mantener.

Tabla 5-14 Programa VHDL estructural para el decodificador de la figura 5-32.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V2to4dec is
  port (I0, I1, EN: in STD_LOGIC;
        Y0, Y1, Y2, Y3: out STD_LOGIC );
end V2to4dec;

architecture V2to4dec_s of V2to4dec is
  signal NOTI0, NOTI1: STD_LOGIC;
  component inv port (I: in STD_LOGIC; O: out STD_LOGIC ); end component;
  component and3 port (I0, I1, I2: in STD_LOGIC; O: out STD_LOGIC ); end component;
begin
  U1: inv port map (I0,NOTI0);
  U2: inv port map (I1,NOTI1);
  U3: and3 port map (NOTI0,NOTI1,EN,Y0);
  U4: and3 port map (   I0,NOTI1,EN,Y1);
  U5: and3 port map (NOTI0,   I1,EN,Y2);
  U6: and3 port map (   I0,   I1,EN,Y3);
end V2to4dec_s;
```

Tabla 5-15 Programa VHDL estilo flujo de datos para un decodificador binario de 3 a 8 tipo 74x138.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V74x138 is
    port (G1, G2A_L, G2B_L: in STD_LOGIC;       -- entradas de habilitación
        A: in STD_LOGIC_VECTOR (2 downto 0);    -- entradas seleccionadas
        Y_L: out STD_LOGIC_VECTOR (0 to 7) );   -- salidas decodificadas
 end V74x138;

architecture V74x138_a of V74x138 is
  signal Y_L_i: STD_LOGIC_VECTOR (0 to 7);
begin
  with A select Y_L_i <=
    "01111111" when "000",
    "10111111" when "001",
    "11011111" when "010",
    "11101111" when "011",
    "11110111" when "100",
    "11111011" when "101",
    "11111101" when "110",
    "11111110" when "111",
    "11111111" when others;
  Y_L <= Y_L_i when (G1 and not G2A_L and not G2B_L)='1' else "11111111";
end V74x138_a;
```

Tabla 5-16 Arquitectura VHDL con un enfoque de mantenimiento para manipulación de nivel activo.

```
architecture V74x138_b of V74x138 is
  signal G2A, G2B: STD_LOGIC;                  -- active-high version of inputs
  signal Y: STD_LOGIC_VECTOR (0 to 7);         -- active-high version of outputs
  signal Y_s: STD_LOGIC_VECTOR (0 to 7);       -- internal signal
begin
  G2A <= not G2A_L; -- convert inputs
  G2B <= not G2B_L; -- convert inputs
  Y_L <= not Y;     -- convert outpouts
  with A select Y_s <=
    "10000000" when "000",
    "01000000" when "001",
    "00100000" when "010",
    "00010000" when "011",
    "00001000" when "100",
    "00000100" when "101",
    "00000010" when "110",
    "00000001" when "111",
    "00000000" when others;
  Y <= Y_s when (G1 and G2A and G2B) = '1' else "00000000";
end V74x138_b;
```

La tabla 5-16 muestra un enfoque de este tipo. No se hacen cambios a las declaraciones `entity`. Sin embargo, las versiones activas altas de las terminales externas activas bajas son definidas dentro de la arquitectura `V74x138_a`, y se emplean proposiciones de asignación explícitas para la conversión entre las señales activas altas y activas bajas. La función misma del decodificador está definida en términos de sólo las señales activas altas, probablemente la mayor ventaja de este enfoque. Otra ventaja es que el diseño puede ser fácilmente modificado precisamente en unos cuantos lugares bien definidos si se requieren cambios en los niveles activos externos.

EJECUCIÓN FUERA DE ORDEN

En la tabla 5-16 hemos agrupado las tres instrucciones de conversión de nivel activo en conjunto al principio del programa; aún así no se asigna un valor a `Y_L` hasta *después* de haber asignado uno a `Y`. Recuerde que esto está bien porque las instrucciones de asignación en la arquitectura se ejecutan en forma concurrente. Es decir, una asignación a cualquier señal provoca que las demás instrucciones que utilizan la señal sean reevaluadas, sin tener en cuenta sus posiciones en el cuerpo de la arquitectura.

Se podría colocar la instrucción "`Y_L <= Y`" al final del cuerpo si le disgusta su posición actual, pero el programa es un poco más sencillo en su forma actual, con todas las conversiones de nivel activo juntas.

Tabla 5-17 Definición jerárquica del decodificador tipo 74x138 con manipulación de nivel activo.

```
architecture V74x138_c of V74x138 is
  signal G2A, G2B: STD_LOGIC;                    -- active-high version of inputs
  signal Y: STD_LOGIC_VECTOR (0 to 7);           -- active-high version of outputs
  component V3to8dec port (G1, G2, G3: in STD_LOGIC);
                          A: in STD_LOGIC_VECTOR (2 downto 0);
                          Y: out STD_LOGIC_VECTOR (0 to 7) );  end component;
begin
  G2A <= not G2A_L;     -- convert inputs
  G2B <= not G2B_L;     -- convert inputs
  Y_L <= not Y;         -- convert outputs
  U1; V3to8dec port map (G1, G2A, G2B, A, Y);
end V74x138_c;
```

Los niveles activos pueden manejarse todavía en una forma más estructurada. Como se muestra en la tabla 5-17, la arquitectura `V74x138` puede ser definida de manera jerárquica, utilizando un componente completamente activo alto `V3to8dec` que tiene su propia definición de estilo de flujo de datos en la tabla 5-18. Una vez más, no se requieren cambios en la definición de alto nivel de la entidad `V74x138`. La figura 5-43 ilustra la relación entre las entidades.

Otro enfoque más para diseño de decodificador se muestra en la tabla 5-19, el cual puede reemplazar la arquitectura `V3to8dec_a` de la tabla 5-18. En lugar de proposiciones

Tabla 5-18
Definición de flujo de datos de un decodificador de 3 a 8 activo alto.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V3to8dec is
    port (G1, G2, G3: in STD_LOGIC;
          A: in STD_LOGIC_VECTOR (2 downto 0);
          Y: out STD_LOGIC_VECTOR (0 to 7) );
end V3to8dec;

architecture V3to8dec_a of V3to8dec is
  signal Y_s: STD_LOGIC_VECTOR (0 to 7);
begin
    with A select Y_s <=
      "10000000" when "000",
      "01000000" when "001",
      "00100000" when "010",
      "00010000" when "011",
      "00001000" when "100",
      "00000100" when "101",
      "00000010" when "110",
      "00000001" when "111",
      "00000000" when others;
    Y <= Y_s when (G1 and G2 and G3)='1'
              else "00000000";
end V3to8dec_a;
```

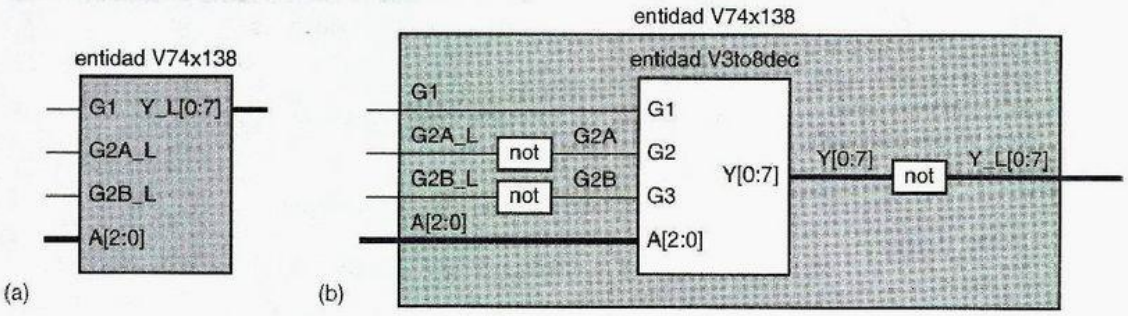


Figura 5-43 Entidad VHDL V74x138; (a) nivel superior; (b) estructura interna usando arquitectura V74x138_c.

COINCIDENCIA DE NOMBRES

En la figura 5-43, los nombres de los puertos de una entidad se dibujan dentro del cuadro correspondiente. Los nombres de las señales que se conectan a los puertos (cuando se utiliza la entidad) se dibujan en las líneas de señal. Note que los nombres de señal pueden coincidir, pero las señales no tienen que hacerlo. El compilador VHDL mantiene todo correcto, asociando un ámbito con cada nombre. La situación es completamente análoga a la manera en que los nombres de parámetro y variable se manejan en lenguajes de programación estructurados, como el C.

concurrentes, esta arquitectura utiliza un proceso y proposiciones secuenciales para definir la operación del decodificador en un estilo de comportamiento. Sin embargo, una estrecha comparación de las dos arquitecturas muestra que en realidad no son tan diferentes excepto por la sintaxis.

```
architecture V3to8dec_b of V3to8dec is
  signal Y_s: STD_LOGIC_VECTOR (0 to 7);
begin
process (A, G1, G2, G3, Y_s)
  begin
    case A is
      when "000" => Y_s <= "10000000";
      when "001" => Y_s <= "01000000";
      when "010" => Y_s <= "00100000";
      when "011" => Y_s <= "00010000";
      when "100" => Y_s <= "00001000";
      when "101" => Y_s <= "00000100";
      when "110" => Y_s <= "00000010";
      when "111" => Y_s <= "00000001";
      when others => Y_s <= "00000000";
    end case;
    if (G1 and G2 and G3) = '1' then Y <= Y_s;
    else <= "00000000";
    end
  end process;
end V3to8dec_b;
```

Tabla 5-19
Definición de arquitectura estilo de comportamiento para un decodificador de 3 a 8.

Tabla 5-20 Definición de arquitectura verdaderamente de comportamiento para un decodificador de 3 a 8.

```
architecture V3to8dec_c of V3to8dec is
begin
process (G1, G2, G3, A)
  variable i: INTEGER range 0 to 7;
  begin
    Y <= "00000000";
    if (G1 and G2 and G3) = '1' then
      for i in 0 to 7 loop
        if i=CONV_INTEGER(A) then Y(i) <= '1'; end if;
      end loop;
    end if;
  end process;
end V3to8dec_c;
```

Como ejemplo final, una arquitectura más de comportamiento en verdad, basada en procesos, para el decodificador de 3 a 8 se muestra en la tabla 5-20. (Recuerde que la función CONV_INTEGER fue definida en la sección 4.7.4.) De los ejemplos que hemos dado, éste es el único que describe la función del decodificador sin incrustar esencialmente una tabla de verdad en el programa VHDL. A ese respecto, es más flexible porque puede ser fácilmente adaptado para hacer un decodificador binario de cualquier tamaño. Al otro respecto, es menos flexible en el sentido que no tiene una tabla de verdad que pueda ser fácilmente modificada para hacer decodificadores personalizados como el que especificamos en la tabla 5-11.

*5.4.8 Decodificadores de siete segmentos

pantalla de siete segmentos

Observe su muñeca y probablemente verá una *pantalla de siete segmentos*. Este tipo de pantalla, que utiliza normalmente diodos emisores de luz (LED, light-emitting diodes) o elementos de pantalla de cristal líquido (LCD, liquid-crystal display), se utiliza en relojes, calculadoras e instrumentos para exhibir datos decimales. Un dígito se exhibe iluminando un subconjunto de los siete segmentos de línea mostrados en la figura 5-44(a).

decodificador de siete segmentos

74x49

Un *decodificador de siete segmentos* tiene el BCD de 4 bits como su código de entrada y el "código de siete segmentos", el cual se describe gráficamente en la figura 5-44(b), como su código de salida. La figura 5-45 y la tabla 5-21 son el diagrama lógico y la tabla de verdad para un decodificador de siete segmentos *74x49*. Excepto por la extraña (¿astuta?) conexión de la entrada de blanqueo BI_L, cada salida del 74x49 es una realización de producto de sumas mínimo para el segmento correspondiente, suponiendo elementos "sin

Figura 5-44 Pantalla de siete segmentos: (a) identificación de segmento; (b) dígitos decimales.

(a) (b)

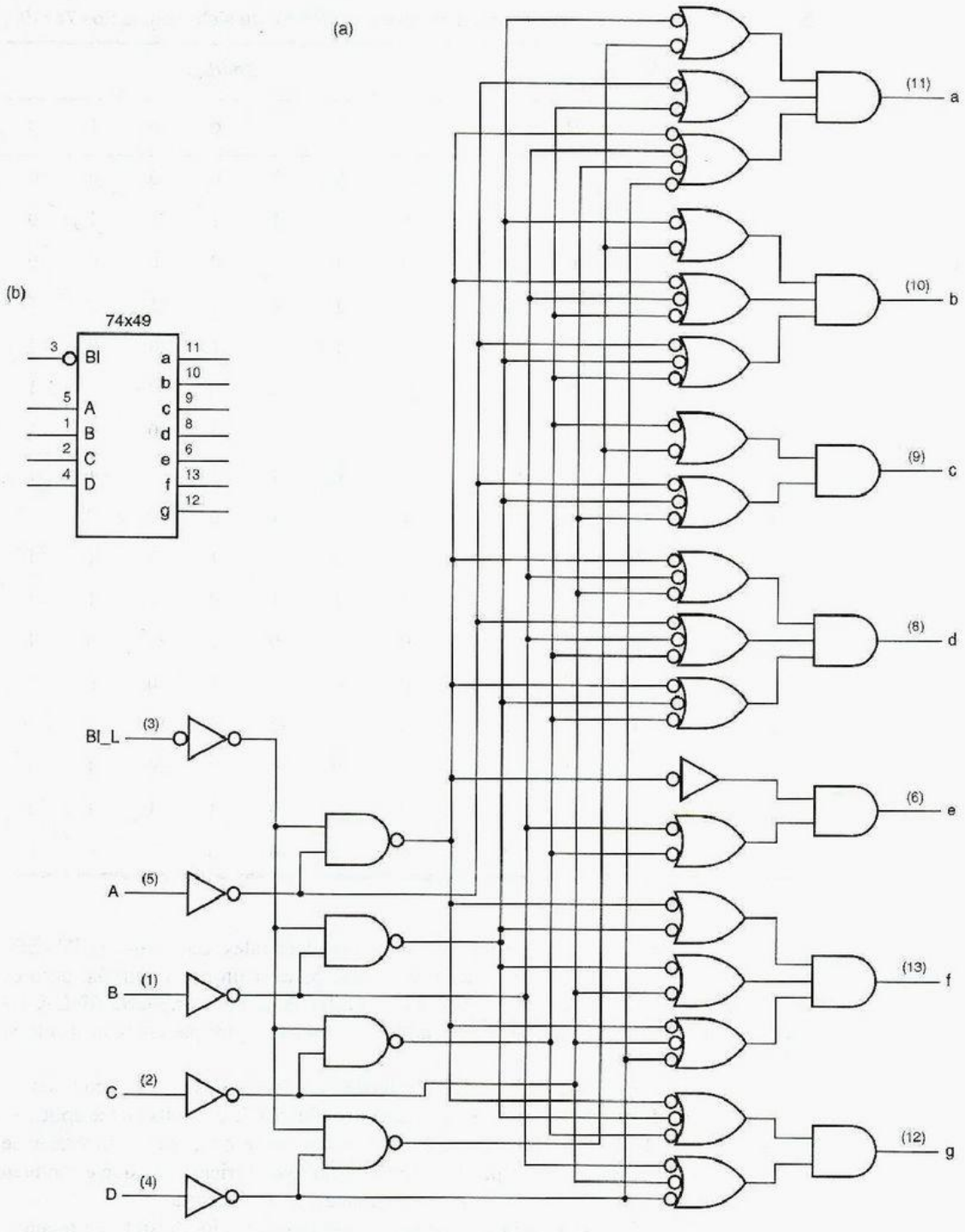


Figura 5-45 El decodificador de siete segmentos 74x49: (a) diagrama lógico, incluyendo números de terminales; (b) símbolo lógico tradicional.

Tabla 5-21 Tabla de verdad para un decodificador de siete segmentos 74x49.

Entradas					*Salidas*						
BI_L	D	C	B	A	a	b	c	d	e	f	g
0	x	x	x	x	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
1	0	0	1	0	1	1	0	1	1	0	1
1	0	0	1	1	1	1	1	1	0	0	1
1	0	1	0	0	0	1	1	0	0	1	1
1	0	1	0	1	1	0	1	1	0	1	1
1	0	1	1	0	0	0	1	1	1	1	1
1	0	1	1	1	1	1	1	0	0	0	0
1	1	0	0	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	0	0	1	1
1	1	0	1	0	0	0	0	1	1	0	1
1	1	0	1	1	0	0	1	1	0	0	1
1	1	1	0	0	0	1	0	0	0	1	1
1	1	1	0	1	1	0	0	1	0	1	1
1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0

importancia" para las combinaciones de entrada no decimales. La estructura INVERSOR-OR-AND utilizada para cada salida puede parecer un poco extraña, pero es equivalente bajo el teorema de DeMorgan generalizado a una compuerta AND-OR-INVERSOR, la cual es una estructura compacta y bastante rápida para ser construida en CMOS o TTL.

Los elementos más modernos de pantalla de siete segmentos tienen decodificadores construidos dentro de ellos, de modo que una palabra BCD de 4 bits puede aplicarse directamente al dispositivo. Muchos de los más antiguos decodificadores discretos de siete segmentos tienen salidas especiales de voltaje alto o corriente alta que están bien elegidas para controlar elementos de pantalla grandes, de alta energía.

La tabla 5-22 es un programa ABEL para un decodificador de siete segmentos. Los conjuntos se utilizan para definir los patrones de dígitos para hacer el programa más legible.

Tabla 5-22 Programa ABEL para un decodificador de siete segmentos tipo 74x49.

```
module Z74X49H
title 'Seven-Segment_Decoder
J. Wakerly, Micro Design Resources, Inc.'
Z74X49H device 'P16L8';

" Input pins
A, B, C, D                  pin 1, 2, 3, 4;
BI_L                        pin 5;
" Output pins
SEGA, SEGB, SEGC, SEGD      pin 19, 18, 17, 16 istype 'com';
SEGE, SEGF, SEGG            pin 15, 14, 13     istype 'com';

" Definitions
BI = !BI_L
DIGITIN = [D,C,B,A];
SEGOUT = [SEGA,SEGB,SEGC,SEGD,SEGE,SEGF,SEGG];

" Segment encodings for digits
DIG0 = [1,1,1,1,1,1,0];  " 0
DIG1 = [0,1,1,0,0,0,0];  " 1
DIG2 = [1,1,0,1,1,0,1];  " 2
DIG3 = [1,1,1,1,0,0,1];  " 3
DIG4 = [0,1,1,0,0,1,1];  " 4
DIG5 = [1,0,1,1,0,1,1];  " 5
DIG6 = [1,0,1,1,1,1,1];  " 6  'tail' included
DIG7 = [1,1,1,0,0,0,0];  " 7
DIG8 = [1,1,1,1,1,1,1];  " 8
DIG9 = [1,1,1,1,0,1,1];  " 9  'tail' included
DIGA = [1,1,1,0,1,1,1];  " A
DIGB = [0,0,1,1,1,1,1];  " b
DIGC = [1,0,0,1,1,1,0];  " C
DIGD = [0,1,1,1,1,0,1];  " d
DIGE = [1,0,0,1,1,1,1];  " E
DIGF = [1,0,0,0,1,1,1];  " F

equations

SEGOUT = !BI & ( (DIGITIN ==  0) & DIG0 # (DIGITIN ==  1) & DIG1
               # (DIGITIN ==  2) & DIG2 # (DIGITIN ==  3) & DIG3
               # (DIGITIN ==  4) & DIG4 # (DIGITIN ==  5) & DIG5
               # (DIGITIN ==  6) & DIG6 # (DIGITIN ==  7) & DIG7
               # (DIGITIN ==  8) & DIG8 # (DIGITIN ==  9) & DIG9
               # (DIGITIN == 10) & DIGA # (DIGITIN == 11) & DIGB
               # (DIGITIN == 12) & DIGC # (DIGITIN == 13) & DIGD
               # (DIGITIN == 14) & DIGE # (DIGITIN == 15) & DIGF );

end Z74X49H
```

5.5 Codificadores

codificador

Un código de salida del decodificador normalmente tiene más bits que su código de entrada. Si el código de salida del dispositivo tiene *menos* bits que el código de entrada, el dispositivo por lo regular se conoce como un *codificador*. Por ejemplo, considere un dispositivo con ocho bits de entrada representando un número binario sin signo, y dos bits de salida indicando si el número es primo o divisible entre 7. Podemos llamar a ese dispositivo un codificador afortunado/primo.

codificador 2^n a n

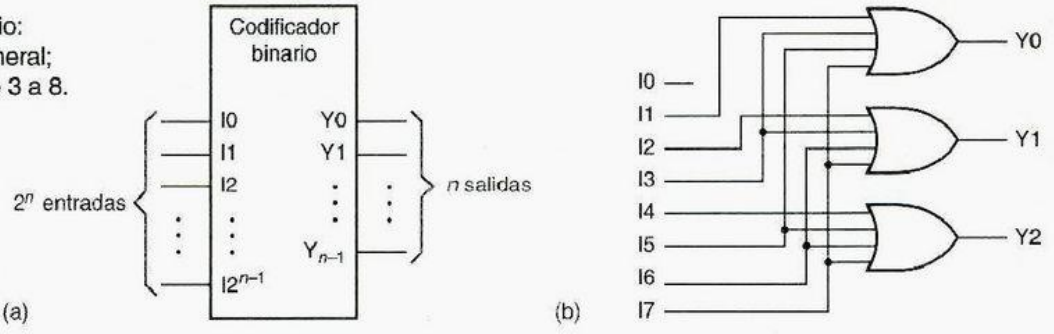
decodificador binario

Probablemente el codificador más simple de construir es un *codificador binario* o de 2^n a n. Como se muestra en la figura 5-46(a), tiene precisamente la función contraria a la de un *de*codificador *binario*: su código de entrada es el código 1 de 2^n y su código de salida es el binario de n bits. Las ecuaciones para un codificador de 8 a 3 con entradas I0-I7 y salidas Y0-Y2 se proporcionan a continuación:

$$Y0 = I1 + I3 + I5 + I7$$
$$Y1 = I2 + I3 + I6 + I7$$
$$Y2 = I4 + I5 + I6 + I7$$

El circuito lógico correspondiente se ilustra en (b). En general, un codificador de 2^n a n puede construirse a partir de n compuertas OR de 2^{n-1} entradas. El bit i del código de entrada está conectado a la j compuerta OR si el bit j en la representación binaria de i es 1.

Figura 5-46
Codificador binario:
(a) estructura general;
(b) codificador de 3 a 8.



5.5.1 Codificadores de prioridad

Las salidas codificadas de 1 de 2^n de un decodificador binario de n bits se utilizan generalmente para controlar un conjunto de 2^n dispositivos, donde como máximo se supone que un dispositivo se encuentra activo en cualquier momento. Ahora bien, considere un sistema con 2^n *entradas*, cada una de las cuales indica una solicitud de servicio, como en la figura 5-47. Esta estructura se encuentra a menudo en subsistemas de entrada/salida de microprocesadores, donde las entradas pueden ser solicitudes de interrupción.

En esta situación, puede parecer natural hacer uso de un codificador binario del tipo mostrado en la figura 5-46 para observar las entradas e indicar cuál está solicitando servicio en cualquier momento. Sin embargo, este codificador funciona adecuadamente solamente si se garantiza que las entradas sean afirmadas como máximo una a la vez. Si

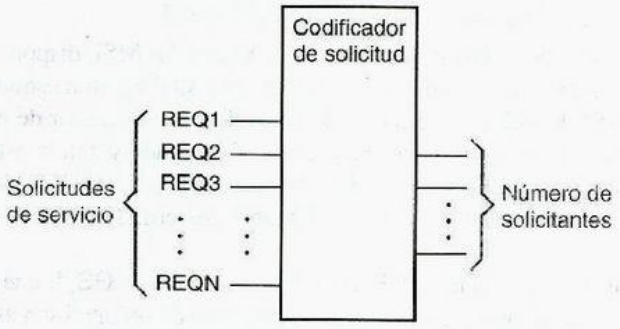


Figura 5-47
Un sistema con 2^n solicitantes, y un "codificador de solicitud" que indica cuál señal de solicitud se encuentra asertiva en cualquier momento.

pueden hacerse varias solicitudes en forma simultánea, el codificador dará resultados indeseables. Por ejemplo, supongamos que las entradas I2 e I4 del codificador de 8 a 3 son ambas 1; entonces la salida es 110, la codificación binaria de 6.

Ya sea 2 o 4, no 6, sería una salida útil en el ejemplo anterior, pero ¿cómo puede decidir el dispositivo de codificación cuál salida es útil? La solución es asignar *prioridad* a las líneas de entrada, de manera que cuando se afirmen múltiples solicitudes, el dispositivo de codificación produzca el número del solicitante de prioridad más alta. Un dispositivo de esta clase se denomina un *codificador de prioridad*.

prioridad

codificador de prioridad

El símbolo lógico para un codificador de prioridad de 8 entradas se ilustra en la figura 5-48. La entrada I7 tiene la prioridad más alta. Las salidas A2–A0 contienen el número de la entrada asertiva de más alta prioridad, si la hay. La salida IDLE es asertiva si ninguna de las entradas es asertiva.

A fin de escribir ecuaciones lógicas para las salidas del codificador de prioridad, primero definimos ocho variables intermedias H0-H7, de modo que Hi es 1 si y sólo si Ii es la entrada I de más alta prioridad:

$$H7 = I7$$
$$H6 = I6 \cdot I7'$$
$$H5 = I5 \cdot I6' \cdot I7'$$
$$\ldots$$
$$H0 = I0 \cdot I1' \cdot I2' \cdot I3' \cdot I4' \cdot I5' \cdot I6' \cdot I7'$$

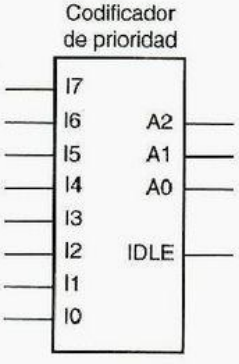


Figura 5-48
Símbolo lógico para un codificador de prioridad genérico de 8 entradas.

Aplicando estas señales, las ecuaciones para las salidas A2-A0 son similares a las de un codificador binario simple:

$$A2 = H4 + H5 + H6 + H7$$
$$A1 = H2 + H3 + H6 + H7$$
$$A0 = H1 + H3 + H5 + H7$$

La salida IDLE es 1 si ninguna entrada es 1:

$$IDLE = (I0 + I1 + I2 + I3 + I4 + I5 + I6 + I7)'$$
$$= I0' \cdot I1' \cdot I2' \cdot I3' \cdot I4' \cdot I5' \cdot I6' \cdot I7'$$

5.5.2 El codificador de prioridad 74x148

74x148

El *74x148* es un codificador de prioridad de 8 entradas MSI, disponible comercialmente. Su símbolo lógico se muestra en la figura 5-49 y su diagrama esquemático se ilustra en la figura 5-50. La principal diferencia entre CI y el codificador de prioridad "genérico" de la figura 5-48 es que sus señales activas de entrada y salida están en estado activo bajo. También, tiene una entrada de habilitación, EI_L, que debe ser asertiva para que cualquiera de sus salidas sea asertiva. La tabla de verdad completa se proporciona en la tabla 5-23.

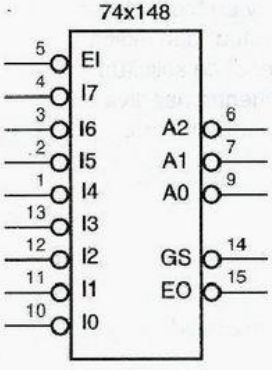


Figura 5-49
Símbolo lógico para el codificador de prioridad 74x148.

En lugar de una salida IDLE, el '148 tiene una salida GS_L que es asertiva cuando el dispositivo es habilitado y una o más de las entradas de solicitud son asertivas. El fabricante llama a esto "selección de grupos" (Group Select, GS), pero es más fácil recordarlo como "algo obtenido" ("Got Something"). La señal EO_L es una *salida* habilitada diseñada para ser conectada a la entrada EI_L de otro '148 que maneja solicitudes de baja prioridad. EO_L es asertiva si EI_L es asertiva pero ninguna entrada de solicitud es asertiva; de este modo, puede habilitarse un '148 de baja prioridad.

La figura 5-51 ilustra cómo cuatro 74x148 pueden conectarse de esta manera para aceptar 32 entradas de solicitudes y producir una salida de 5 bits, RA4-RA0, indicando el solicitante de prioridad más alta. Puesto que las salidas A2-A0 de como máximo un '148 se habilitarán en cualquier momento, las salidas de los '148 individuales pueden funcionar con dispositivos OR para producir RA2-RA0. De igual manera, las salidas individuales GS_L pueden combinarse en un codificador de 4 a 2 para producir RA4 y RA3. La salida RGS es asertiva si cualquier salida GS es asertiva.

Tabla 5-23 Tabla de verdad para un codificador de prioridad de 8 entradas 74x148.

Entradas									*Salidas*				
EI_L	I0_L	I1_L	I2_L	I3_L	I4_L	I5_L	I6_L	I7_L	A2_L	A1_L	A0_L	GS_L	EO_L
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	0	1	0	1
0	x	x	x	x	x	0	1	1	0	1	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	x	0	1	1	1	1	1	1	0	1	0	1
0	x	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

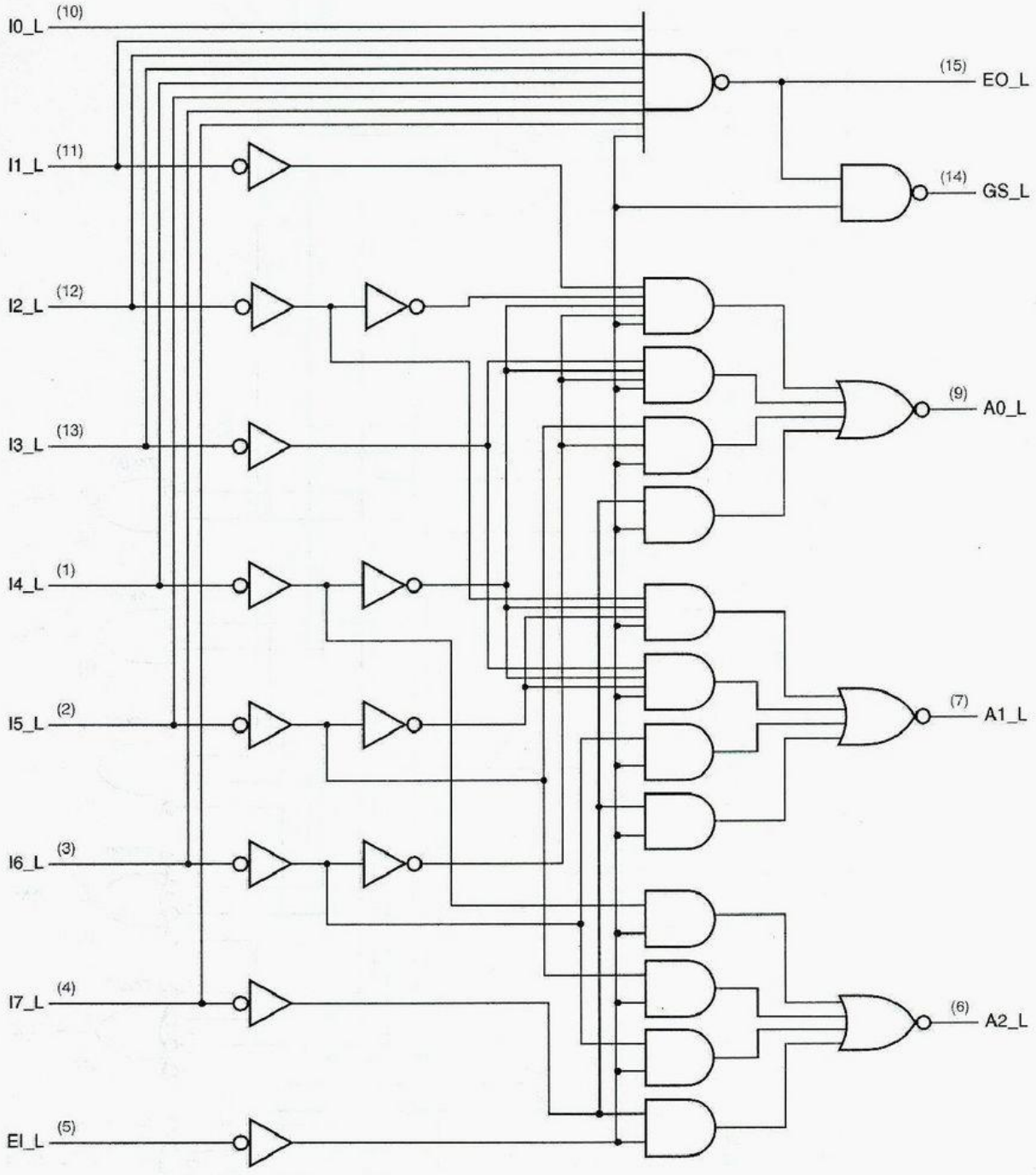


Figura 5-50 Diagrama lógico para el codificador de prioridad de 8 entradas 74x148, incluyendo los números de terminales para un encapsulado en doble línea de 16 terminales estándar.

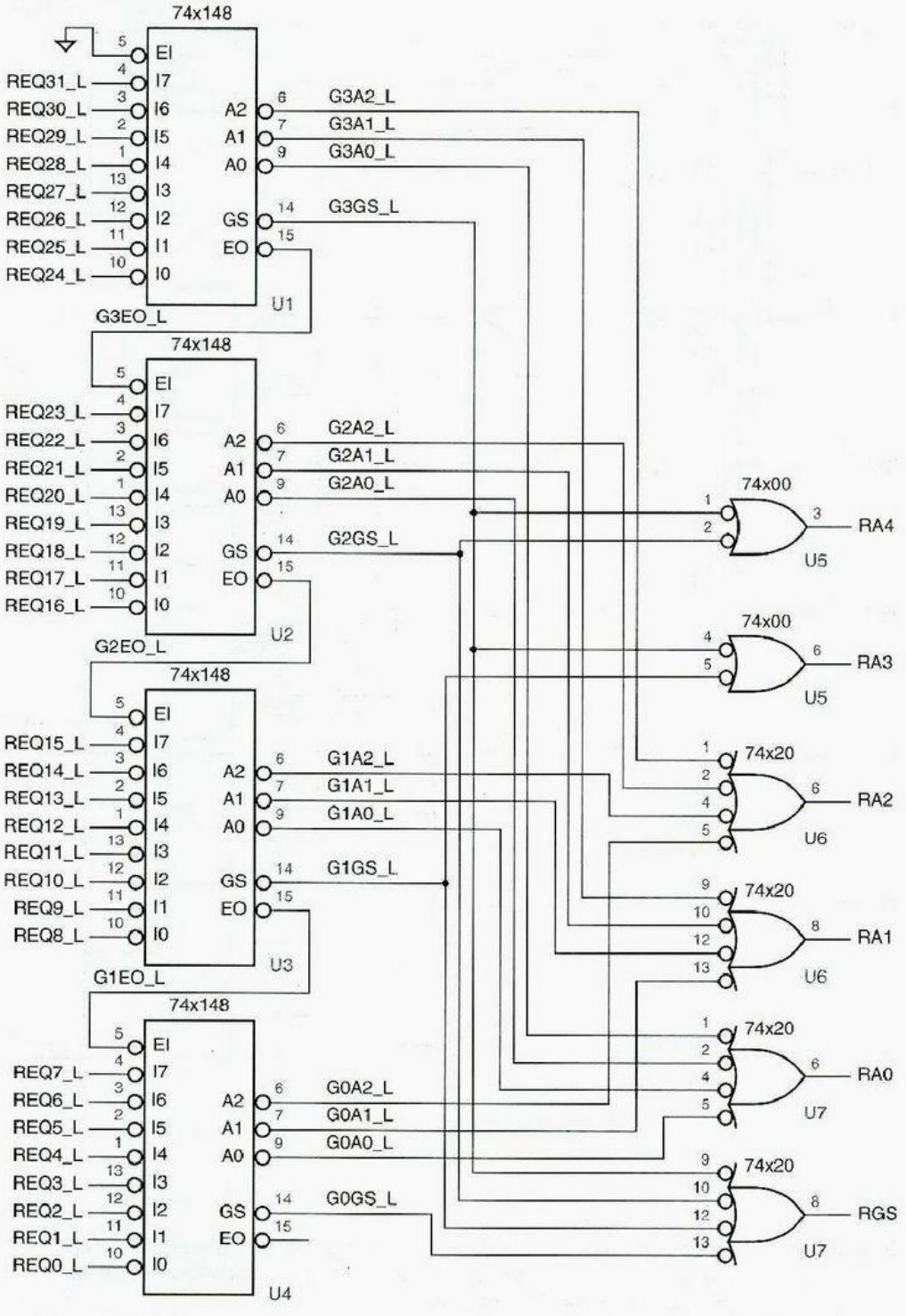


Figura 5-51 Aquí se muestran cuatro 74x148 en cascada para manejar 32 solicitudes.

5.5.3 Codificadores en ABEL y PLD

Los codificadores pueden diseñarse en ABEL utilizando una ecuación explícita para cada una de las combinaciones de entrada, como en la tabla 5-8 de la página 361, o utilizando tablas de verdad. Sin embargo, puesto que el número de entradas es por lo regular grande, el número de combinaciones de entrada es muy grande, y este método con frecuencia no es práctico.

Por ejemplo, ¿cómo especificaríamos un codificador de prioridad de 15 entradas para entradas P0–P14? ¡Obviamente no queremos tratar con todas las 2^{15} posibles combinaciones de entrada! Una manera de hacer esto es descomponer la función de prioridad en dos partes. En primer lugar, escribimos ecuaciones para 15 variables Hi ($0 \le i \le 14$) de tal manera que Hi es 1 si Pi es la entrada afirmada de más alta prioridad. Puesto que por definición como máximo una variable Hi es 1 en cualquier momento, podemos combinar las Hi en un codificador binario para obtener un número de 4 bits identificando la entrada afirmada de más alta prioridad.

Un programa ABEL que utiliza este enfoque se muestra en la tabla 5-24, y un diagrama lógico para el codificador empleando un solo PAL20L8 o GAL20V8 se proporciona en la figura 5-52. Las entradas P0–P14 son asertivas para indicar solicitudes, teniendo P14 la más alta prioridad. Si EN_L (Enable) es asertiva, entonces las salidas Y3_L–Y0_L proporcionan el número (bajo activo) de la solicitud de más alta prioridad, y GS es asertiva si cualquier solicitud se encuentra presente. Si EN_L está negada, entonces las salidas Y3_L–Y0_L están negadas y GS está negada. ENOUT_L está afirmada si EN_L está afirmada y no se encuentra presente ninguna solicitud.

Nótese que en el programa ABEL, las ecuaciones para las variables Hi son escritas como "expresiones constantes", antes de la declaración `equations`. De esta forma, estas señales no serán generadas explícitamente. Más bien, serán incorporadas en las subsiguientes ecuaciones para Y0-Y3, que el compilador manipula para obtener expresiones de

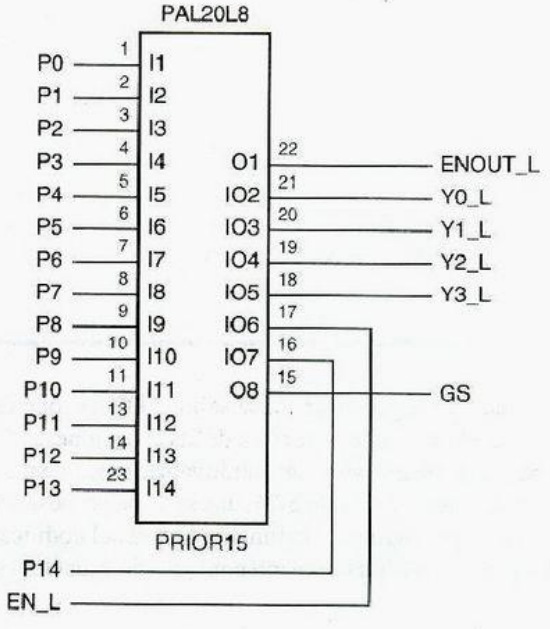


Figura 5-52
Diagrama lógico para un codificador de prioridad de 15 entradas basado en PLD.

Tabla 5-24 Un programa ABEL para un codificador e prioridad de 15 entradas.

```
module PRIOR15
title '15-Input Priority Encoder '
PRIOR15 device 'P20L8';

" Input and output pins
P0..P14, EN_L                          pin 1..11, 13, 14, 23, 16, 17;
Y3_L, Y2_L, Y1_L, Y0_L, GS, ENOUT_L    pin 18..21, 15, 22 istype 'com';

" Active-level translation
EN = !EN_L; ENOUT = !ENOUT_L;
Y3 = !Y3_L; Y2 = !Y2_L; Y1 = !Y1_L; Y0 = !Y0_L;

" Constant expressions
H14 = EN&P14;
H13 = EN&!P14&P13;
H12 = EN&!P14&!P13&P12;
H11 = EN&!P14&!P13&!P12&P11;
H10 = EN&!P14&!P13&!P12&!P11&P10;
H9  = EN&!P14&!P13&!P12&!P11&!P10&P9;
H8  = EN&!P14&!P13&!P12&!P11&!P10&!P9&P8;
H7  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&P7;
H6  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&P6;
H5  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&P5;
H4  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&!P5&P4;
H3  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&!P5&!P4&P3;
H2  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&!P5&!P4&!P3&P2;
H1  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&!P5&!P4&!P3&!P2&P1;
H0  = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&!P5&!P4&!P3&!P2&!P1&P0;

equations

Y3 = H8 # H9 # H10 # H11 # H12 # H13 # H14;
Y2 = H4 # H5 # H6 # H7 # H12 # H13 # H14;
Y1 = H2 # H3 # H6 # H7 # H10 # H11 # H14;
Y0 = H1 # H3 # H5 # H7 # H9 # H11 # H13;

GS    = EN&(P14#P13#P12#P11#P10#P9#P8#P7#P6#P5#P4#P3#P2#P1#P0);
ENOUT = EN&!P14&!P13&!P12&!P11&!P10&!P9&!P8&!P7&!P6&!P5&!P4&!P3&!P2&!P1&!P0;

end PRIOR15
```

suma de productos mínimas. A medida que se producen, cada salida Yi tiene solamente siete términos de producto, como se puede ver de la estructura de las ecuaciones.

El codificador de prioridad puede diseñarse aún más intuitivamente empleando la instrucción WHEN de ABEL. Como se muestra en la tabla 5-25, una serie de proposiciones WHEN profundamente anidadas expresan precisamente la función lógica del codificador de prioridad. Este programa produce con exactitud el mismo conjunto de ecuaciones de salida que el programa anterior.

Tabla 5-25 Programa ABEL alterno para el mismo codificador de prioridad de 15 entradas.

```
module PRIOR15W
title '15-Input Priority Encoder'
PRIOR15W device 'P20L8';

" Input and output pins
P0..P14, EN_L                          pin 1..11, 13, 14, 23, 16, 17;
Y3_L, Y2_L, Y1_L, Y0_L, GS, ENOUT_L    pin 18..21, 15, 22 istype 'com';

" Active-level translation
EN = !EN_L; ENOUT = !ENOUT_L;

" Set definition
Y_L = [Y3_L, Y2_L, Y1_L, Y0_L];

equations

WHEN !EN THEN !Y_L = 0;              " Note: !Y_L === active-high Y
ELSE WHEN P14 THEN !Y_L = 14;        " Can't define active-high Y set
  ELSE WHEN P13 THEN !Y_L = 13;      "       due to ABEL set quirks
    ELSE WHEN P12 THEN !Y_L = 12;
      ELSE WHEN P11 THEN !Y_L = 11;
        ELSE WHEN P10 THEN !Y_L = 10;
          ELSE WHEN P9 THEN !Y_L = 9;
            ELSE WHEN P8 THEN !Y_L = 8;
              ELSE WHEN P7 THEN !Y_L = 7;
                ELSE WHEN P6 THEN !Y_L = 6;
                  ELSE WHEN P5 THEN !Y_L = 5;
                    ELSE WHEN P4 THEN !Y_L = 4;
                      ELSE WHEN P3 THEN !Y_L = 3;
                        ELSE WHEN P2 THEN !Y_L = 2;
                          ELSE WHEN P1 THEN !Y_L = 1;
                            ELSE WHEN P0 THEN !Y_L = 0;
                              ELSE {!Y_L = 0; ENOUT = 1;};

GS = EN&(P14#P13#P12#P11#P10#P9#P8#P7#P6#P5#P4#P3#P2#P1#P0);

end PRIOR15W
```

5.5.4 Codificadores en VHDL

El enfoque para la especificación de codificadores en VHDL es semejante al enfoque de ABEL. Podríamos incrustar el equivalente de una tabla de verdad o ecuaciones explícitas en el programa VHDL, pero una descripción de comportamiento es bastante más intuitiva. Puesto que la construcción IF-THEN-ELSE de VHDL describe mejor la acción de dar prioridad y está disponible solamente con un proceso, utilizamos el enfoque de comportamiento basado en procesos.

La tabla 5-26 es un programa de comportamiento VHDL para un codificador de prioridad cuya función es equivalente al 74x148. Hace uso de un ciclo FOR para buscar

Tabla 5-26 Programa VHDL de comportamiento para un codificador de prioridad de 8 entradas tipo 74x148.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V74x148 is
    port (
        EI_L: in STD_LOGIC;
        I_L: in STD_LOGIC_VECTOR (7 downto 0);
        A_L: out STD_LOGIC_VECTOR (2 downto 0);
        EO_L, GS_L: out STD_LOGIC
    );
end V74x148;

architecture V74x148p of V74x148 is
  signal EI: STD_LOGIC;                          -- active-high version input
  signal I: STD_LOGIC_VECTOR (7 downto 0);       -- active-high version of inputs
  signal EO, GS: STD_LOGIC;                      -- active-high version of outputs
  signal A: STD_LOGIC_VECTOR (2 downto 0);       -- active-high version of outputs
begin
  process (EI_L, I_L, EI, EO, GS, I, A)
  variable j: INTEGER range 7 downto 0;
  begin
    EI <= not EI_L; -- convierte entrada
    I <= not I_L;   -- convierte entradas
    EO <= '1'; GS <= '0'; A <= "000";
    if (EI)= '0' then EO <= '0';
    else for j in 7 downto 0 loop
        if I(j)='1' then
          GS <= '1'; EO <= '0'; A <= CONV_STD_LOGIC_VECTOR(j,3);
          exit;
        end if;
      end loop;
    end if;
    EO_L <= not EO; -conver output
    GS_L <= not GS; - convert output
    A_L <= not A; - convert outputs
  end process;
end V74x148p;
```

una entrada, comenzando con la entrada de más alta prioridad. Como algunos de nuestros programas anteriores, realiza explícitamente la conversión de nivel activo al principio y al final. También recuerde que la función `CONV_STD_LOGIC_VECTOR(j,n)` fue definida en la sección 4.7.4 para convertir de un entero `j` a un `STD_LOGIC_VECTOR` de una longitud especificada `n`. Este programa se modifica fácilmente para utilizar un orden de prioridad diferente o un número distinto de entradas o para agregar más funcionalidad tal como para hallar una segunda entrada de más alta prioridad, como se explora en la sección 6.2.3.

5.6 Dispositivos de tres estados

En las secciones 3.7.3 y 3.10.5 describimos el diseño eléctrico de dispositivos CMOS y TTL cuyas salidas pueden estar en uno de tres estados: 0, 1 o Hi-Z. En esta sección mostraremos cómo hacer uso de éstos.

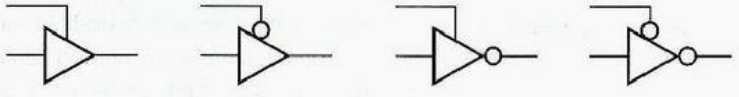
5.6.1 Buffers de tres estados

buffer de tres estados
controlador de tres estados

El dispositivo más básico de tres estados es un *buffer (o controlador) de tres estados*. Los símbolos lógicos para cuatro buffers de tres estados que son físicamente diferentes se ilustran en la figura 5-53. El símbolo básico es el de un buffer no inversor [(a), (b)] o un inversor [(c), (d)]. La señal extra en la parte superior del símbolo es una entrada de *habilitación de tres estados*, la cual puede ser activa alta [(a), (c)] o activa alta [(b), (d)]. Cuando la entrada de habilitación se encuentra afirmada, el dispositivo se comporta como un buffer ordinario o inversor. Cuando la entrada de habilitación está negada, la salida del dispositivo "flota", es decir, se va a un estado desconectado de alta impedancia (Hi-Z) y funcionalmente se comporta como si no estuviera allí.

Los dispositivos de tres estados permiten que múltiples fuentes compartan una sola "línea de reunión", mientras que solamente un dispositivo "dialoga" en la línea a la vez. La figura 5-54 nos proporciona un ejemplo de cómo puede hacerse esto. Tres bits de entrada SSRC2–SSRC0, seleccionan una de ocho fuentes de datos que pueden controlar una sola lína, SDATA. Un decodificador de 3 a 8, el 74x138, asegura que sólo una de las ocho líneas SEL sea asertiva a la vez, habilitando únicamente un buffer de tres estados para controlar SDATA. Sin embargo, si no todas las líneas EN se encuentran asertivas, entonces ninguno de los buffers, de tres estados es habilitado. El valor lógico en SDATA se encuentra indefinido en este caso.

Figura 5-53 Varios buffers de tres estados: (a) no invertido, habilitación activa alta; (b) no invertido, habilitación activa baja; (c) invertido, habilitación activa alta, (d) invertido, habilitación activa baja.



DEFINICIÓN DE "INDEFINIDO"

El nivel de voltaje real de una señal flotante depende de varios factores del circuito, tales como carga capacitiva y resistiva; y puede variar en el tiempo. Además, la interpretación de este nivel por parte de otros circuitos depende de las características de entrada de estos circuitos. Por esto es aconsejable tratar a una señal flotante como si fuera "indefinida". En ocasiones se emplea un resistor de pull-up en líneas de reunión de tres estados para asegurar que un valor flotante sea llevado a un voltaje ALTO e interpretado como un 1 lógico. Esto es especialmente importante en líneas de reunión que controlen dispositivos CMOS, los cuales pueden consumir corriente en exceso, cuando su voltaje de entrada se encuentra a la mitad del camino entre el 0 y el 1 lógicos.

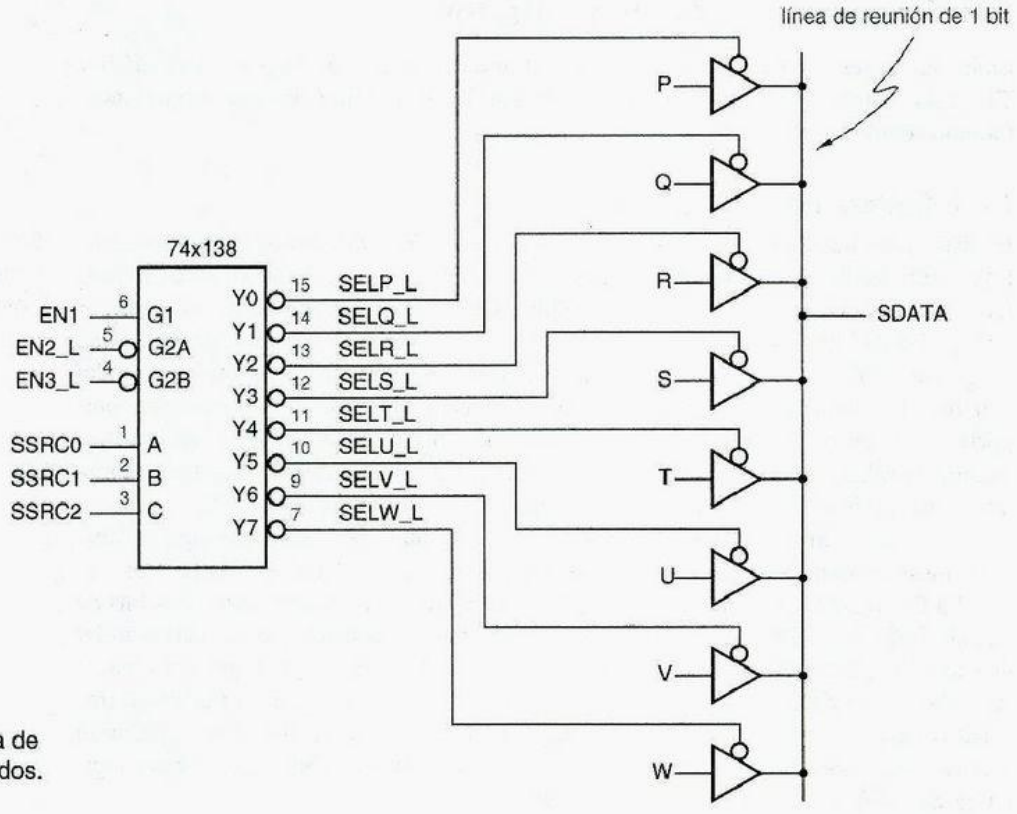


Figura 5-54
Ocho fuentes que comparten una línea de reunión de tres estados.

Los dispositivos típicos de tres estados están diseñados de modo que entren al estado Hi-Z más rápido de lo que salen del mismo. (En términos de las especificaciones en un libro de datos, t_{pLZ} y t_{pHZ} son ambos menores que t_{pZL} y t_{pZH}; véase también la sección 3.7.3.) Esto significa que si las salidas de dos dispositivos de tres estados están conectadas a la misma línea de reunión, y simultáneamente deshabilitamos una y habilitamos la otra, el primer dispositivo se saldrá de la línea de reunión antes de que la segunda se ponga en contacto. Esto es importante porque, si ambos dispositivos fueran a controlar la línea de reunión al mismo tiempo, y si ambos estuvieran intentando mantener valores de salida opuestos (0 y 1), entonces fluiría corriente excesiva y crearía ruido en el sistema, como lo discutimos en la sección 3.7.7. Esto se conoce a menudo como *confrontación*.

confrontación

Por desgracia, los retardos y sesgos de temporización en los circuitos de control hacen difícil asegurar que las entradas de habilitación de diferentes dispositivos de tres estados cambien "simultáneamente". Incluso cuando esto es posible, surge un problema si los dispositivos de tres estados de familias lógicas con velocidades diferentes (o inclusive diferentes CI fabricados en distintos días) se conectan a la misma línea de reunión. El tiempo de encendido (t_{pZL} o t_{pZH}) de un dispositivo "rápido" puede ser más breve que el tiempo de apagado (t_{pLZ} o t_{pHZ}) de uno "lento", y las salidas todavía pueden estar en conflicto.

La única manera realmente segura de emplear dispositivos de tres estados es diseñar una lógica de control que garantice un *tiempo muerto* en la línea de reunión,

tiempo muerto

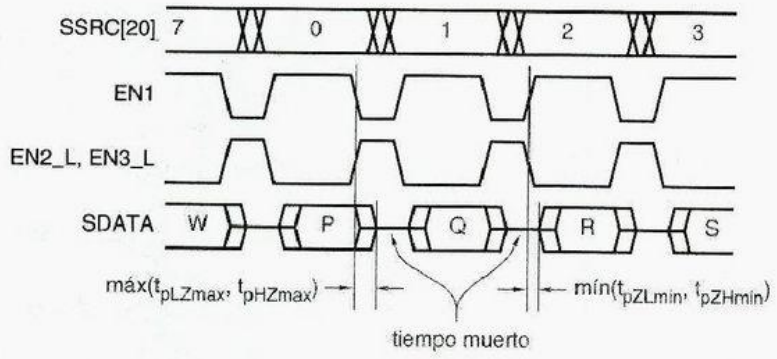


Figura 5-55
Diagrama de temporización para la línea de reunión de tres estados.

durante el cual ninguno la esté controlando. El tiempo muerto debe ser suficientemente largo para dar cuenta de las diferencias, en el peor de los casos, entre los tiempos de apagado y encendido de los dispositivos y de los sesgos en las señales de control de tres estados. Un diagrama de temporización que ilustra esta clase de operación para la línea de reunión de la figura 5-54 se muestra en la figura 5-55. Este diagrama de temporización también ilustra una convención de dibujo para señales de tres estados: cuando se encuentra en el estado Hi-Z, se exhiben como un nivel "indefinido" entre 0 y 1.

5.6.2 Buffers de tres estados SSI y MSI estándar

Como las compuertas lógicas, se pueden incluir diversos buffers de tres estados independientes pueden en un solo encapsulado de CI tipo SSI. Por ejemplo, la figura 5-56 ilustra las terminales externas del *74x125* y del *74x126*, cada uno de los cuales contiene cuatro buffers de tres estados no inversores independientes en un encapsulado de 14 terminales. Las entradas de habilitación de tres estados en el '125 son activas bajas y en el '126 son activas altas.

74x125
74x126

La mayor parte de las aplicaciones de línea de reunión utilizan un bus con más de un bit de datos. Por ejemplo, en un sistema microprocesador de 8 bits, el bus de datos es de ocho bits de ancho, y los dispositivos periféricos normalmente sitúan los datos en el bus ocho bits a la vez. De este modo, un dispositivo periférico habilita ocho controladores de tres estados para controlar el bus, todos al mismo tiempo. Las entradas de habilitación independientes, como en el '125 y el '126, no son necesarias.

De esta forma, para reducir el tamaño del encapsulado en las aplicaciones de bus ancho, la mayoría de las partes MSI comúnmente utilizadas contienen múltiples buffers de tres estados con entradas de habilitación comunes. Por ejemplo, la figura 5-57 muestra el diagrama lógico y el símbolo para un separador *74x541* de tres estados no

74x541

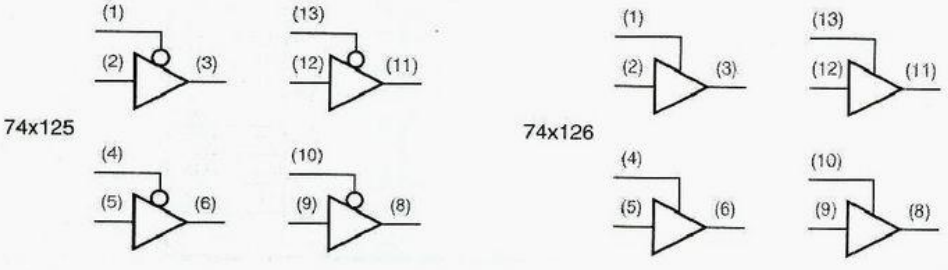


Figura 5-56
Terminales externas de los buffers de tres estados 74x125 y 74x126.

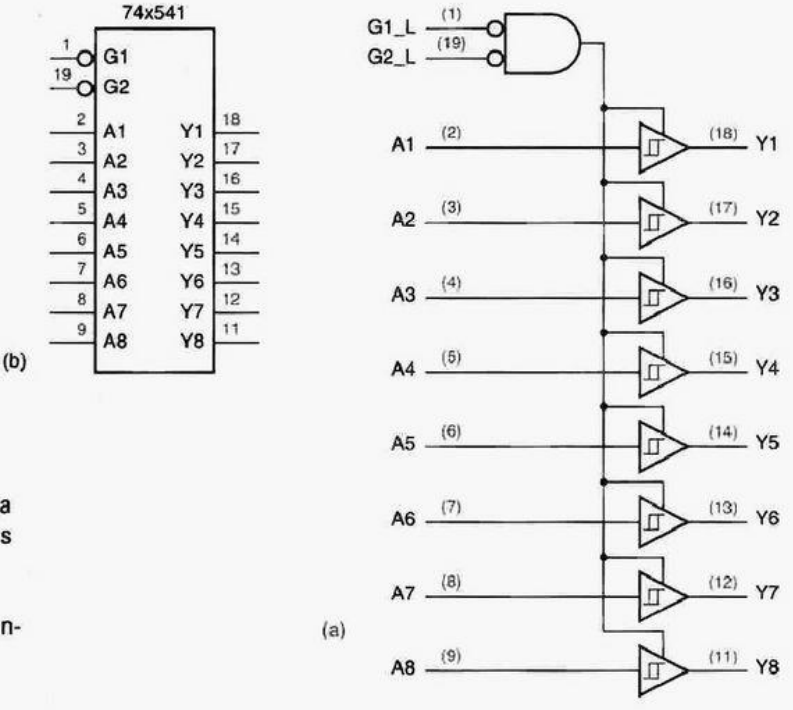


Figura 5-57
El buffer octal de tres estados 74x541: (a) diagrama lógico, incluyendo números de terminal para un encapsulado en doble línea de 20 terminales estándar; (b) símbolo lógico tradicional.

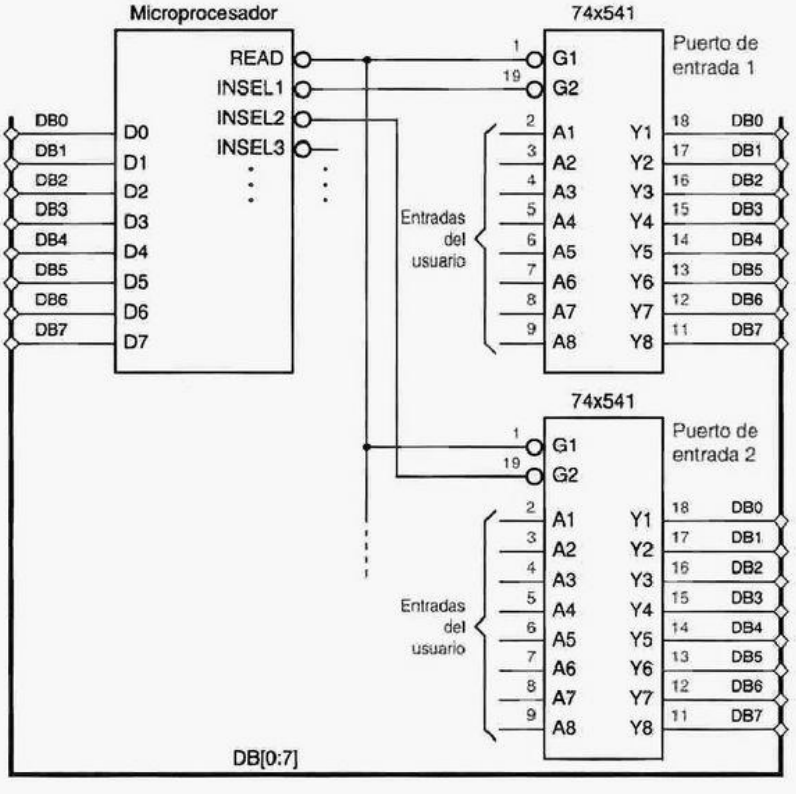
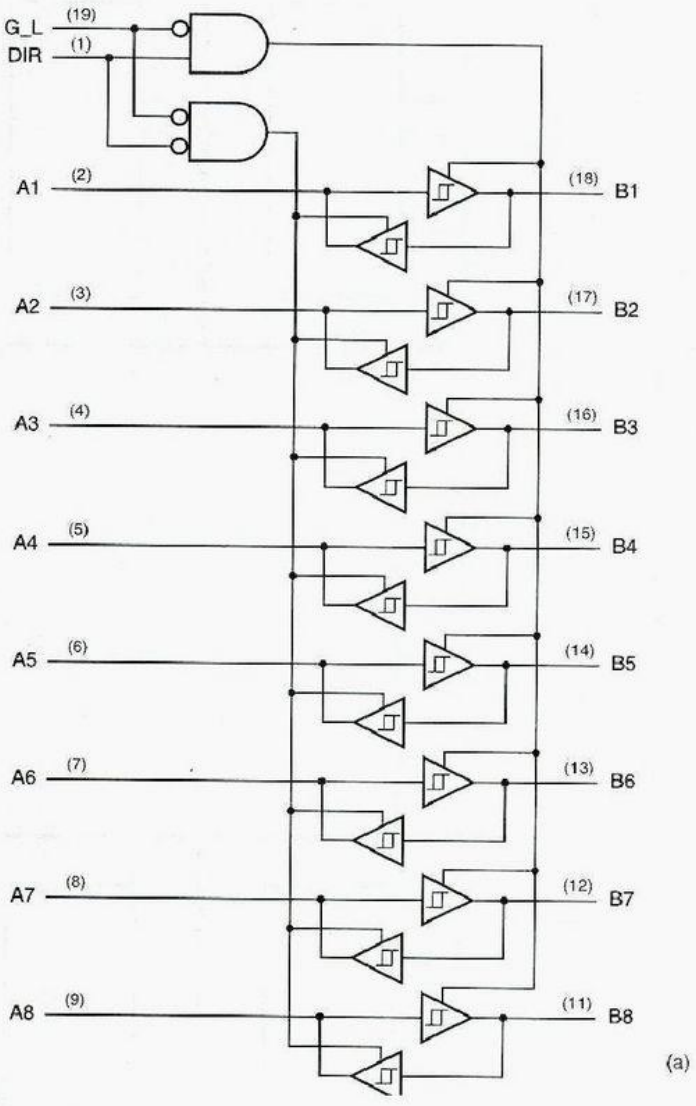


Figura 5-58
Empleo de un 74x541 como un puerto de entrada de microprocesador.



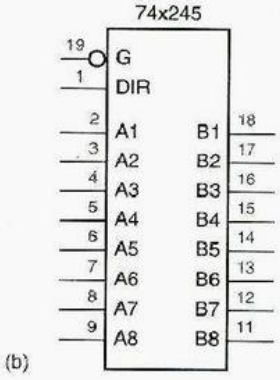


Figura 5-59
El transceptor octal de tres estados 74x245: (a) diagrama lógico; (b) símbolo lógico tradicional.

inversor octal. *Octal* quiere decir que la parte contiene ocho buffers individuales. Ambas entradas de habilitación, G1_L y G2_L, deben estar afirmadas para habilitar las salidas de tres estados del dispositivo. Los pequeños símbolos rectangulares dentro de los símbolos de separador indican *histéresis*, una característica eléctrica de las entradas que mejoran la inmunidad al ruido, como lo explicamos en la sección 3.7.2. Las entradas del 74x541 por lo regular tienen 0.4 volts de histéresis.

octal

histéresis

La figura 5-58 muestra parte de un sistema microprocesador con un bus de datos de 8 bits, DB[0–7], y un 74x541 que se utiliza como puerto de entrada. El microprocesador selecciona el Puerto de Entrada 1 al afirmar INSEL1 y solicitar una operación de lectura mediante la aserción de READ. El 74x541 responde al controlar el bus de datos del microprocesador con datos de entrada suministrados por el usuario. Pueden seleccionarse otros puertos de entrada cuando se afirma una línea INSEL diferente junto con READ.

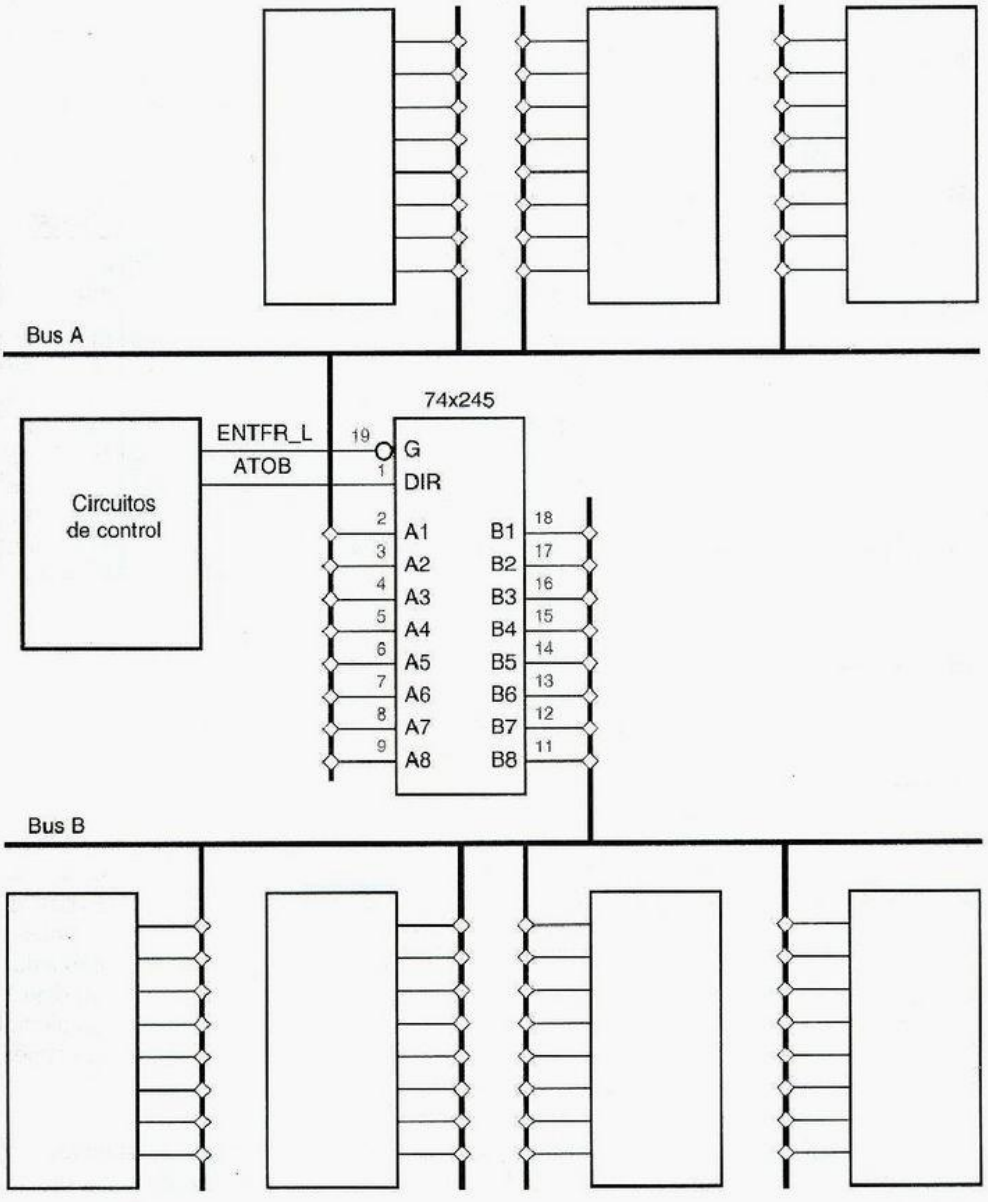


Figura 5-60
Buses bidireccionales y operación del transceptor.

74x540
74x240
74x241

Muchas otras variedades de buffers de tres estados octales se encuentran disponibles comercialmente. Por ejemplo, el *74x540* es idéntico al 74x541 excepto en que contiene buffers inversores. El *74x240* y el *74x241* son semejantes al '540 y al '541, excepto que aquéllos se encuentran divididos en dos secciones de 4 bits, cada uno con una sola línea de habilitación.

transceptor de bus

Un *transceptor de bus* contiene pares de buffers de tres estados conectados en direcciones opuestas entre cada par de terminales de manera que los datos pueden transferirse en cualquier dirección. Por ejemplo, la figura 5-59 de la página anterior ilustra el

74x245

diagrama lógico y el símbolo para un transceptor octal de tres estados *74x245*. La entrada

Tabla 5-27 Modos de operación para un par de buses bidireccionales.

ENTFR_L	ATOB	*Operación*
0	0	Datos de transferencia desde una fuente en el bus B hasta un destino en el bus A.
0	1	Datos de transferencia desde una fuente en el bus A hasta un destino en el bus B.
1	x	Datos de transferencia en los buses A y B en forma independiente.

DIR determina la dirección de transferencia, desde A hasta B (DIR = 1) o de B hacia A (DIR = 0). El buffer de tres estados para la dirección seleccionada se habilita solamente si G_L es asertiva.

Un transceptor de bus por lo regular es utilizado entre dos *buses bidireccionales*, como se muestra en la figura 5-60. Son posibles tres diferentes modos de operación, dependiendo del estado de G_L y DIR, como se muestra en la tabla 5-27. Como es habitual, es la responsabilidad del diseñador asegurar que ningún bus sea controlado en algún momento por dos dispositivos. Sin embargo, pueden presentarse transferencias independientes donde ambos buses sean controlados al mismo tiempo, cuando el transceptor está deshabilitado, como se indica en el último renglón de la tabla.

bus bidireccional

5.6.3 Salidas de tres estados en ABEL y PLD

Las aplicaciones PLD combinacionales en las secciones anteriores han empleado las terminales de E/S bidireccionales (IO2-1O7 en un PAL16L8 o GAL16V8) de manera estática, es decir, siempre habilitadas o siempre deshabilitadas de salida. En tales aplicaciones, el compilador puede tener cuidado de programar las compuertas de habilitación de salida apropiadamente: todos los fusibles fundidos o todos intactos. De manera predeterminada en ABEL, una terminal de salida de tres estados se programa para estar siempre habilitada si su nombre de señal aparece en el lado izquierdo de una ecuación, y siempre deshabilitada en caso contrario.

Las terminales de salida de tres estados también se pueden controlar en forma dinámica, mediante una sola entrada, mediante un término de producto o, empleando lógica de dos pasos, mediante una expresión lógica más compleja. En ABEL, se vincula un *sufijo de atributo* `.OE` a un nombre de señal en el lado izquierdo de una ecuación para indicar que la ecuación se aplica a la habilitación de salida para la señal. En un PAL16L8 o GAL16V8, la habilitación de salida está controlada por una compuerta AND simple, de modo que el lado derecho de la ecuación de habilitación debe reducirse a un solo término de producto.

sufijo de atributo `.OE`

La tabla 5-28 muestra un simple fragmento de programa PLD con control de tres estados. Adaptado del programa para un decodificador tipo 74x138 en la tabla 5-8, este programa incluye un control OE de salida de tres estados para todas las ocho salidas del decodificador. Nótese que se define un conjunto `Y` para permitir que todas las ocho habilitaciones de salida se especifiquen en una sola ecuación; el sufijo `.OE` se aplica a cada miembro del conjunto.

En el ejemplo anterior, las terminales de salida Y0–Y7 se encuentran siempre habilitadas o flotando, y se utilizan estrictamente como "terminales de salida". Las terminales de E/S (IO2–IO7 en un 16L8 o 16V8) pueden emplearse como "terminales bidireccionales";

Tabla 5-28 Programa ABEL para un decodificador binario de 3 a 8 tipo 74x138 con control de salida de tres estados.

```
module Z74X138T
title '74x138 Decoder with Three-State Output Enable'
Z74X138T device 'P16L8';

" Input pins
A, B, C, !G2A, !G2B, G1, !OE  pin 1, 2, 3, 4, 5, 6, 7;
" Output pins
!Y0, !Y1, !Y2, !Y3            pin 19, 18, 17, 16 istype 'com';
!Y4, !Y5, !Y6, !Y7            pin 15, 14, 13, 12 istype 'com';

" Constant expression
ENB = G1 & G2A & G2B;
Y = [Y0..Y7];

equations
Y.OE = OE;
Y0 = ENB & !C & !B & !A;
...
Y7 = ENB &  C &  B &  A;

end Z74X138T
```

es decir, se pueden utilizar en forma dinámicamente como entradas o salidas dependiendo de si la compuerta de habilitación de salida está produciendo un 0 o un 1. Una aplicación de ejemplo de las terminales de E/S es un transceptor de bus de 2 bits y cuatro vías con las especificaciones siguientes:

- El transceptor maneja cuatro buses bidireccionales de 2 bits, A[1:2], B[1:2], C[1:2] y D[1:2].
- La fuente de datos para controlar los buses se selecciona mediante tres entradas de selección, S[2:0], de acuerdo con la tabla 5-29. Si S2 es 0, los buses se controlan con un valor constante, de otro modo son controlados por uno de los otros buses. Sin embargo, cuando la fuente seleccionada es un bus, el bus fuente se controla con 00.

Tabla 5-29 Códigos de selección de bus para un transceptor de bus de cuatro vías.

S2	S1	S0	*Fuente seleccionada*
0	0	0	00
0	0	1	01
0	1	0	10
0	1	1	11
1	0	0	Bus A
1	0	1	Bus B
1	1	0	Bus C
1	1	1	Bus D

Tabla 5-30 Un programa ABEL para un transceptor de bus de 2 bits y cuatro vías.

```
module XCVR4X2
title 'Four-way 2-bit Bus Transceiver'
XCVR4X2 device 'P16L8';

" Input pins
A1I, A2I                          pin 1, 11;
!AOE, !BOE, !COE, !DOE, !MOE      pin 2, 3, 4, 5, 6;
S0, S1, S2                        pin 7, 8, 9;
" Output and bidirectional pins
A1O, A2O                          pin 19, 12                  istype 'com';
B1, B2, C1, C2, D1, D2            pin 18, 17, 16, 15, 14, 13 istype 'com';

" Set definitions
ABUSO = [A1O,A2O];
ABUSI = [A1I,A2I];
BBUS  = [B1,B2];
CBUS  = [C1,C2];
DBUS  = [D1,D2];
SEL   = [S2,S1,S0];
CONST = [S1,S0];
" Constants
SELA = [1,0,0];
SELB = [1,0,1];
SELC = [1,1,0];
SELD = [1,1,1];

equations
ABUSO.OE = AOE & MOE;
BBUS.OE = BOE & MOE;
CBUS.OE = COE & MOE;
DBUS.OE = DOE & MOE;
ABUSO = !S2&CONST # (SEL==SELB)&BBUS # (SEL==SELC)&CBUS # (SEL==SELD)&DBUS;
BBUS = !S2&CONST # (SEL==SELA)&ABUSI # (SEL==SELC)&CBUS # (SEL==SELD)&DBUS;
CBUS = !S2&CONST # (SEL==SELA)&ABUSI # (SEL==SELB)&BBUS # (SEL==SELD)&DBUS;
DBUS = !S2&CONST # (SEL==SELA)&ABUSI # (SEL==SELB)&BBUS # (SEL==SELC)&CBUS;

end XCVR4X2
```

- Cada bus tiene su propia señal de habilitación de salida, AOE_L, BOE_L, COE_L o DOE_L. También existe una señal de habilitación de salida "maestra", MOE_L. El transceptor controla un bus particular si y sólo si MOE_L y la señal de habilitación de salida para ese bus se encuentran afirmadas en conjunto.

La tabla 5-30 es un programa ABEL que realiza la función del transceptor. De acuerdo con las ecuaciones (.OE) de habilitación, cada bus está habilitado a la salida si MOE y su propia OE están afirmadas. Cada bus está controlado con S1 y S0 si S2 es 0, y con el

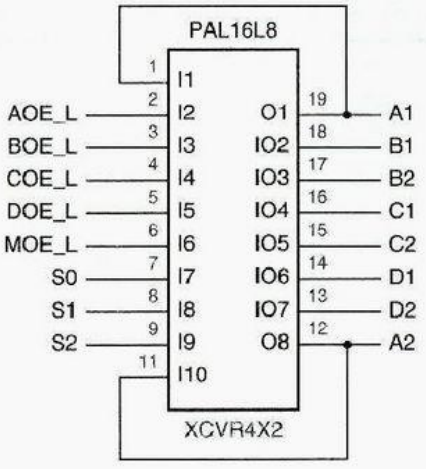


Figura 5-61
Entradas y salidas PLD para un transceptor de bus de 2 bits y cuatro vías.

bus seleccionado si se elige un bus diferente. Si el mismo bus es elegido, la ecuación de salida se evalúa a 0, y el bus es controlado con 00 como es requerido.

La figura 5-61 es un diagrama lógico para un PAL16L8 (o un GAL16V8) con las entradas y salidas requeridas. Puesto que el dispositivo tiene solamente seis terminales bidireccionales y la especificación requiere ocho, el bus A utiliza un par de terminales para entrada y otro para salida. Esto se refleja en el programa mediante el uso de señales separadas y conjuntos para la entrada y salida del bus A.

Tabla 5-31 Declaraciones del paquete IEEE 1164 para `STD_ULOGIC` y `STD_LOGIC`.

```
PACKAGE std_logic_1164 IS
-- logic state system (unresolved)
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing Unknown
                         '0',  -- Forcing 0
                         '1',  -- Forcing 1
                         'Z',  -- High Impedance
                         'W',  -- Weak Unknown
                         'L',  -- Weak
                         'H',  -- Weak
                         '-',  -- Don't care
                       );

-- unconstrained array of std_ulogic
   TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

-- resolution function
   FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;

-- *** industry standard logic type
   SUBTYPE std_logic IS resolved stdd_ulogic;
...
```

Tabla 5-32 Cuerpo del paquete IEEE 1164 para STD_ULOGIC y STD_LOGIC.

```
PACKAGE BODY std_logic_1164 IS
-- local type
    TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

-- resolution function
    CONSTANT resolution_table : stdlogic_table := (
    --        ---------------------------------------------------
    --        |  U    X    0    1    Z    W    L    H    -      |  |
    --        ---------------------------------------------------
              ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
              ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
              ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
              ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
              ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
              ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
              ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
              ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
              ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | - |
        );

    FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
        VARIABLE result : std_ulogic := 'Z'; --Weahest state default
    BEGIN
        -- the test for a single driver is essential other wise the
        -- loop would return 'X' for a single driver of '-' and that
        -- would conflict with the value of a single driver unresolved.
        -- signal
        IF     (s'LENGTH = 1) THEN   RETURN s(s'LOW);
        ELSE
            FOR i IN s'RANGE LOOP
                result := resolution_table(result, s(i));
            END LOOP;
        END IF;
        RETURN result;
    END resolved;
...
```

*5.6.4 Salidas de tres estados en VHDL

VHDL por sí mismo no tiene operadores y tipos integrados para las salidas de tres estados. Sin embargo, tiene primitivas que se pueden utilizar para crear señales y sistemas con comportamiento de tres estados; el paquete IEEE 1164 emplea estas primitivas. Como un comienzo, el tipo STD_LOGIC del IEEE 1164 define 'Z' como uno de sus nueve posibles valores de señal; este valor es empleado para el estado de alta impedancia. Usted puede asignar este valor a cualquier señal STD_LOGIC, y las definiciones de las funciones lógicas estándar consideran la posibilidad de entradas 'Z' (generalmente una entrada 'Z' provocará una salida 'U').

Dada la disponibilidad de las señales de tres estados, ¿cómo creamos buses de tres estados en VHDL? Un bus de tres estados generalmente tiene dos o más controladores, aunque los mecanismos que discutimos funcionan bien con solamente un controlador. En VHDL, no hay construcción de lenguaje explícita para unir salidas de tres estados en un bus. En cambio, el compilador automáticamente junta señales que son controladas en dos o más procesos diferentes, es decir, las señales que aparecen en el lado izquierdo de una instrucción de asignación de señal en dos o más procesos. Sin embargo, las señales deben tener el tipo apropiado, como se explica a continuación.

subtipo resuelto
STD_ULOGIC
tipo resuelto
función de resolución

El tipo STD_LOGIC IEEE 1164 está definido en realidad como un subtipo "resuelto" de un tipo sin resolver; *STD_ULOGIC*. En VHDL, un *tipo resuelto* se utiliza para cualquier señal que pueda ser controlada en dos o más procesos. La definición de un tipo resuelto incluye una *función de resolución* que es llamada cada vez que se hace una asignación a una señal que tiene ese tipo. Como su nombre lo indica, esta función resuelve el valor de la señal cuando tiene múltiples controladores.

Las tablas 5-31 y 5-32 de las páginas anteriores muestran las definiciones IEEE 1164 de STD_ULOGIC, STD_LOGIC y la función de resolución "resolved". Este código utiliza un arreglo bidimensional resolution_table para determinar el valor final de STD_LOGIC producido por *n* procesos que controlan una señal para *n* valores posiblemente diferentes pasados en el vector de entrada s. Si, por ejemplo, una señal tiene cuatro controladores, el compilador de VHDL automáticamente construye un vector de 4 elementos con los cuatro valores de controlador, y pasa este vector a resolved cada vez que cualquiera de los valores cambia. El resultado se pasa de regreso a la simulación.

Nótese que el orden en el que los valores de señal controlados aparecen en s no afecta el resultado producido por resolved, debido a la fuerte ordenación de "fuerzas" en la resolution_table: 'U'>'X'>'0','1'>'W'>'L','H'>'-'. Es decir, una vez que una señal es parcialmente resuelta para un valor particular, nunca se resuelve adicionalmente para un valor "más débil"; y los conflictos 0/1 y L/H siempre se resuelven para un valor indefinido más fuerte ('X' o 'W').

De este modo, ¿se necesita conocer todo esto a fin de emplear salidas de tres estados en VHDL? Bien, habitualmente no, pero puede ayudar si sus simulaciones no coinciden con la realidad. Todo lo que se requiere normalmente para usar salidas de tres estados dentro de VHDL es declarar las señales correspondientes como de tipo STD_ULOGIC.

Por ejemplo, la tabla 5-33 describe un sistema que utiliza cuatro controladores de tres estados de 8 bits (en cuatro procesos) para seleccionar uno de cuatro buses de 8 bits, A, B, C y D, para manejar un bus de resultado X. Recuerde que la construcción "(others => 'Z')" denota un vector de todas las Z, con cualquier longitud que sea necesaria para satisfacer el lado izquierdo de la asignación.

VHDL es lo suficiente flexible para poder usarlo para definir otros tipos de operación de bus. Por ejemplo, se podría definir un subtipo y función de resolución para salidas de drenaje abierto de tal modo que se obtenga una función AND alambrada. Sin embargo, rara vez se tendrá que utilizar esta característica; las definiciones para tipos de salidas específicas en los PLD, FPGA y ASIC por lo regular ya están hechas en las bibliotecas que proporcionan los proveedores de componentes.

Tabla 5-33 Programa VHDL con cuatro controladores de tres estados de 8 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V3statex is
    port (
        G_L: in STED_LOGIC;                          -- Habilitación de salida global
        SEL: in STED_LOGIC_VECTOR (1 downto 0);      -- Selección de entrada
                                                        0,1,2,3, ==> A,B,C,D
        A, B, C, D: in STD_LOGIC_VECTOR (1 to 8);    -- Buses de entrada
        X: out STD_LOGIC_VECTOR (1 to 8)             -- Bus de salida (tres estados)
    );
end V3statex;

architecture V3states of V3states is
begin
  process (G_L, SEL, A)
  begin
    if G_L='0' and SEL  = "00" then X <= A;
    else X <= (others => 'Z');
    end if;
  end process;

  process (G_L, SEL, B)
  begin
    if G_L='0' and SEL = "01" then X <= B;
    else X <= (others => 'Z');
    end if;
  end process;

  process (G_L, SEL, C)
  begin
    if G_L='0' and SEL = "10" then X <= C;
    else X <= (others => 'Z');
    end if;
  end process;

  process (G_L, SEL, D)
  begin
    if G_L='0' and SEL = "11" then X <= D;
    else X <= (others => 'Z');
    end if;
  end process;

end V3states;
```

5.7 Multiplexores

multiplexor

Un *multiplexor* es un interruptor digital: conecta los datos desde una de n fuentes hasta su salida. La figura 5.62(a) ilustra las entradas y salidas de un multiplexor de n entradas y b bits. Existen n fuentes de datos, cada una de las cuales es de b bits de ancho, y se tienen b bits de salida. En los multiplexores típicos que están disponibles en el mercado, $n = 1, 2, 4, 8$ o 16, y $b = 1, 2$ o 4. Existen s entradas que seleccionan entre las n fuentes, de modo que $s = \lceil \log_2 n \rceil$. Una entrada de habilitación EN permite al multiplexor "hacer su trabajo"; cuando EN = 0, todas las salidas son 0. A un multiplexor a menudo se le denomina de manera abreviada: *mux*.

mux

La figura 5-62(b) muestra un circuito interruptor que es aproximadamente equivalente al multiplexor. Sin embargo, a diferencia de un interruptor mecánico, un multiplexor es un dispositivo unidireccional; la información fluye solamente desde las entradas (a la izquierda) hacia las salidas (a la derecha).

Podemos escribir una ecuación lógica general para una salida de multiplexor:

$$\text{iY} = \sum_{j=0}^{n-1} \text{EN} \cdot \text{M}_j \cdot \text{iDj}$$

Aquí, el símbolo de sumatoria representa una suma lógica de términos de producto. La variable iY es un bit de salida particular ($1 \le i \le b$), y la variable iDj es el bit i de entrada de la fuente j ($0 \le j \le n-1$). M_j representa el minitérmino j de las s entradas seleccionadas. De este modo, cuando el multiplexor es habilitado y el valor en las entradas seleccionadas es j, cada salida iY es igual al correspondiente bit de la entrada seleccionada, iDj.

Los multiplexores son obviamente dispositivos útiles en cualquier aplicación en la cual los datos deben ser conmutados desde múltiples fuentes hacia un destino. Una aplicación común en computadoras es el multiplexor entre los registros del microprocesador y su unidad aritmética lógica (ALU, por sus siglas en inglés). Por ejemplo, considere un

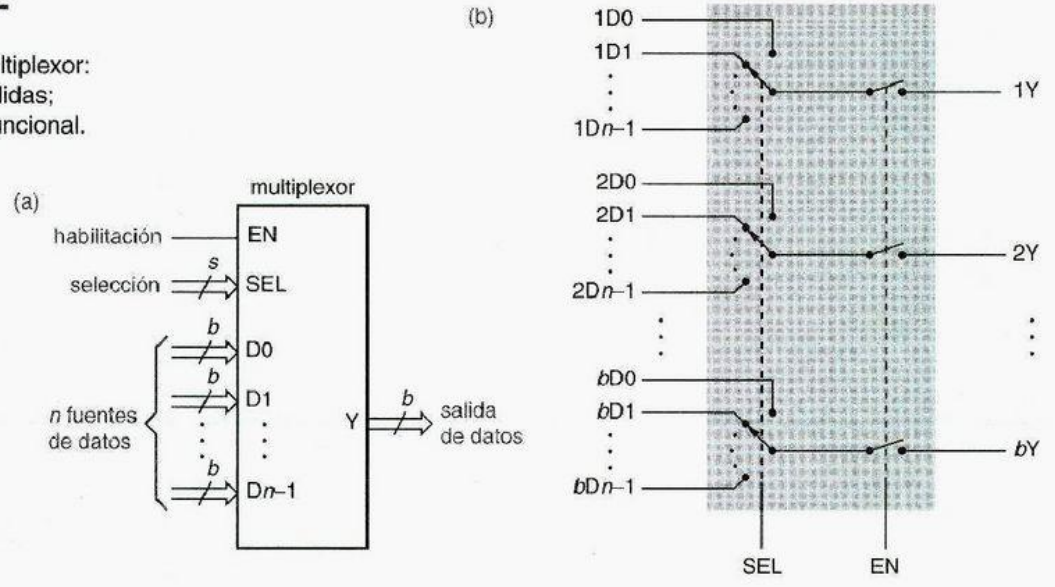


Figura 5-62
Estructura de multiplexor:
(a) entradas y salidas;
(b) equivalente funcional.

procesador de 16 bits en el que cada instrucción tiene un campo de 3 bits que especifica uno de ocho registros por utilizar. Este campo de 3 bits se encuentra conectado a las entradas de selección de un multiplexor de 16 bits y 8 entradas. Las entradas de datos del multiplexor están conectadas a los ocho registros y sus salidas de datos están conectadas a la ALU para ejecutar la instrucción empleando el registro seleccionado.

5.7.1 Multiplexores MSI estándar

Los tamaños de los multiplexores MSI disponibles comercialmente están limitados por el número de terminales disponibles en un encapsulado de CI de bajo precio. Los multiplexores comúnmente empleados vienen en encapsulados de 16 terminales. En un extremo se encuentra el *74x151*, mostrado en la figura 5-63, el cual selecciona entre ocho entradas de 1 bit. Las entradas de selección están nombradas C, B y A donde C es más significativo numéricamente. La entrada de habilitación EN_L es activa baja se suministran tanto versiones de salida activas altas (Y) como activas bajas (Y_L).

74x151

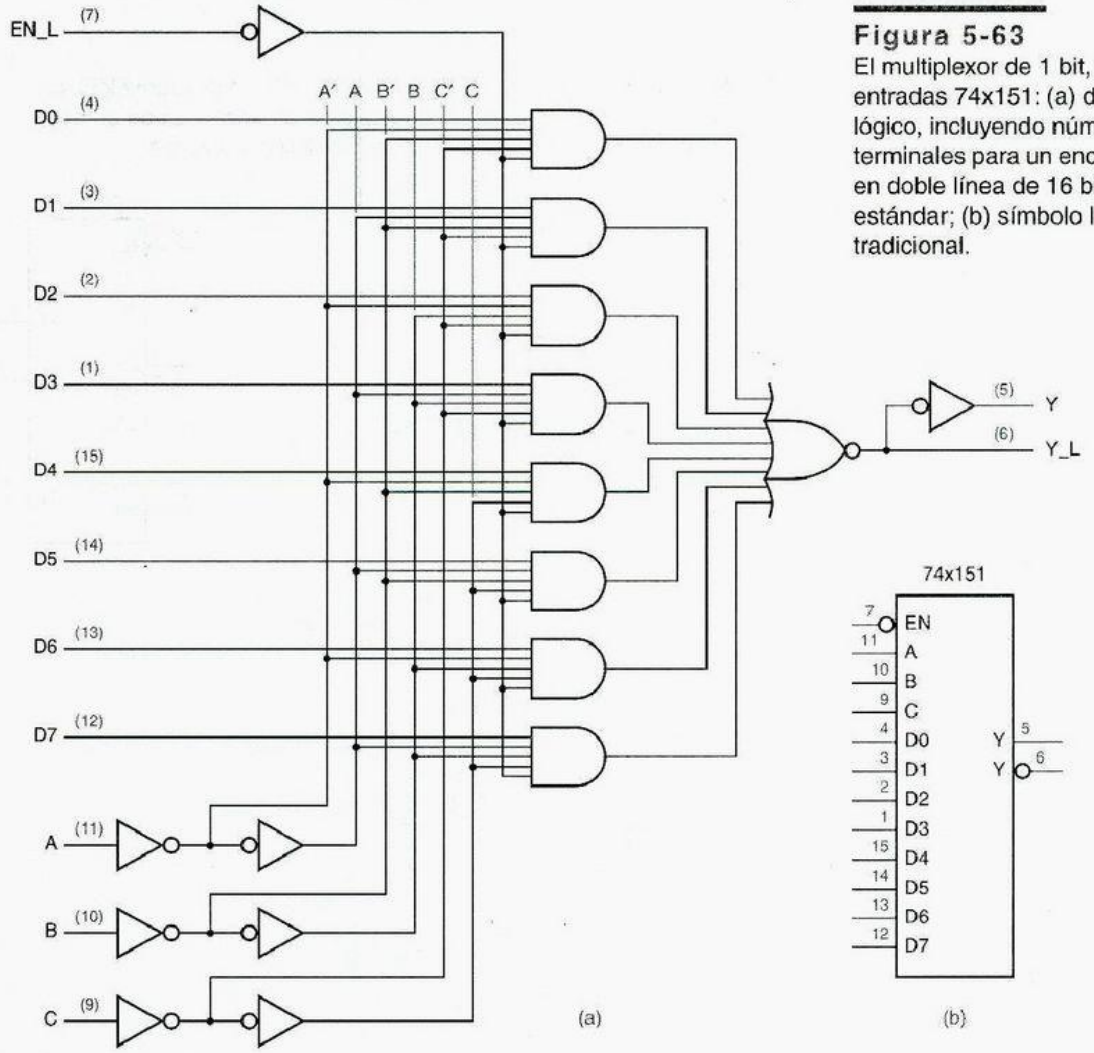
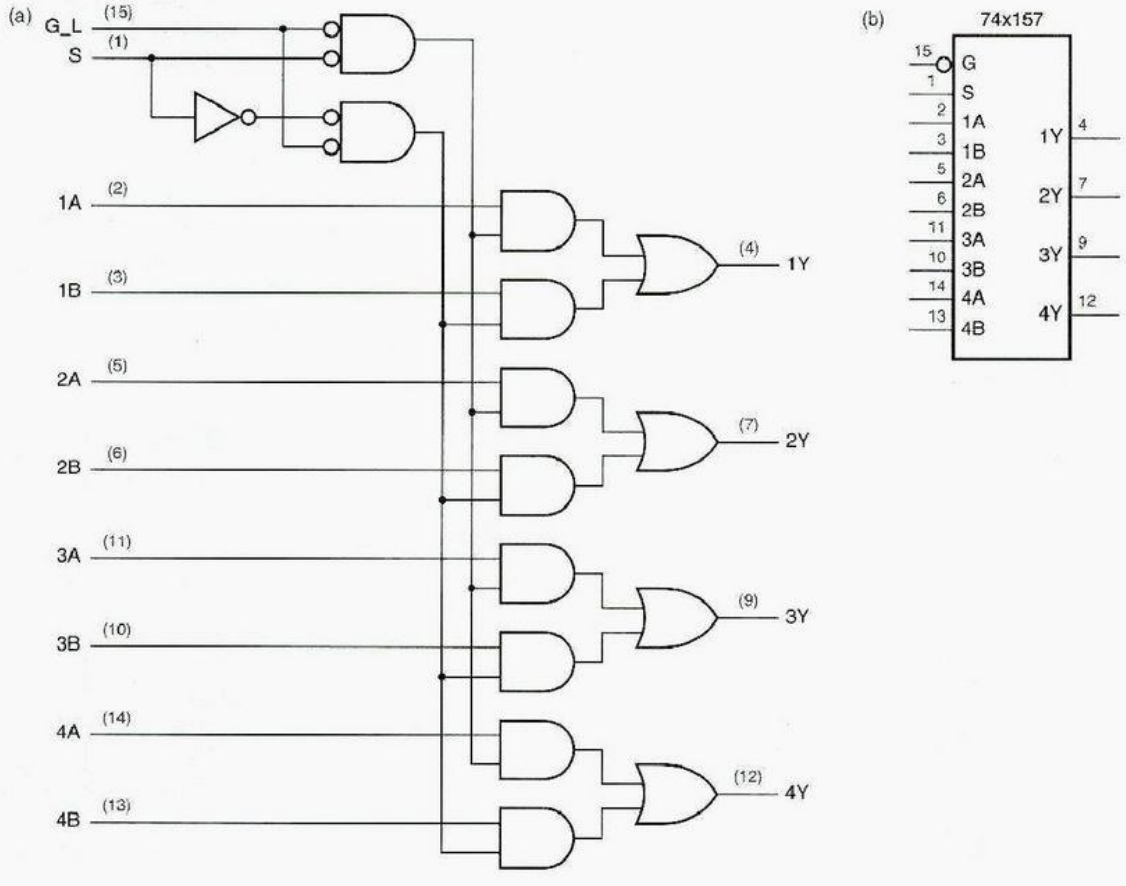


Figura 5-63
El multiplexor de 1 bit, 8 entradas 74x151: (a) diagrama lógico, incluyendo números de terminales para un encapsulado en doble línea de 16 bits estándar; (b) símbolo lógico tradicional.

Tabla 5-34
Tabla de verdad para un multiplexor de 1 bit, 8 entradas 74x151.

Entradas				*Salidas*	
EN_L	C	B	A	Y	Y_L
1	x	x	x	0	1
0	0	0	0	D0	D0′
0	0	0	1	D1	D1′
0	0	1	0	D2	D2′
0	0	1	1	D3	D3′
0	1	0	0	D4	D4′
0	1	0	1	D5	D5′
0	1	1	0	D6	D6′
0	1	1	1	D7	D7′

Figura 5-64 El multiplexor de 4 bits, 2 entradas 74x157: (a) diagrama lógico, incluyendo números de terminales para un encapsulado en doble línea de 16 bits estándar; (b) símbolo lógico tradicional.



Entradas		Salidas			
G_L	S	1Y	2Y	3Y	4Y
1	x	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B

Tabla 5-35
Tabla de verdad para un multiplexor de 4 bits, 2 entradas 74x157.

La tabla de verdad del 74x151 se muestra en la tabla 5-34. Aquí, una vez más, hemos extendido nuestra notación para tablas de verdad. Hasta ahora, las que hemos presentado han especificado una salida de 0 o 1 para cada combinación de entrada. En la tabla del 74x151, solamente algunas de las entradas se enumeran bajo el encabezado de "entradas". Cada salida está especificada como 0, 1 o una función lógica simple de las entradas restantes (por ejemplo, D0 o D0'). Esta notación ahorra ocho columnas y ocho renglones en la tabla y presenta la función lógica con más claridad que la haría una tabla más extensa.

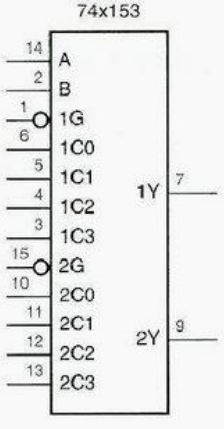
En el otro extremo de los multiplexores en encapsulados de 16 terminales, tenemos el *74x157*, mostrado en la figura 5-64, el cual selecciona entre dos entradas de 4 bits. Tan sólo para confundir las cosas, el fabricante ha nombrado a la entrada de selección S y la entrada de habilitación activa baja G_L. Nótese también que las fuentes de datos son nombradas A y B en lugar de D0 y D1 como en nuestro ejemplo genérico. Nuestra notación de tabla de verdad extendida hace la descripción de los 74x157 muy compacta, como se muestra en la tabla 5-35. *74x157*

Entre el 74x151 y el 74x157 se encuentra el *74x153*, un multiplexor de 2 bits y 4 entradas. Este dispositivo, cuyo símbolo lógico se ilustra en la figura 5-65, tiene entradas de habilitación separadas (1G, 2G) para cada bit. Como se muestra en la tabla 5-36, su función es muy directa. *74x153*

Entradas				Salidas	
1G_L	2G_L	B	A	1Y	2Y
0	0	0	0	1C0	2C0
0	0	0	1	1C1	2C1
0	0	1	0	1C2	2C2
0	0	1	1	1C3	2C3
0	1	0	0	1C0	0
0	1	0	1	1C1	0
0	1	1	0	1C2	0
0	1	1	1	1C3	0
1	0	0	0	0	2C0
1	0	0	1	0	2C1
1	0	1	0	0	2C2
1	0	1	1	0	2C3
1	1	x	x	0	0

Tabla 5-36
Tabla de verdad para un multiplexor de 2 bits, 4 entradas 74x153.

Figura 5-65
Símbolo lógico tradicional para el 74x153.



74x251

74x253

74x257

Algunos multiplexores tienen salidas de tres estados. La entrada de habilitación de un multiplexor de este tipo, en lugar de forzar las salidas a cero, las fuerza al estado Hi-Z. Por ejemplo, el *74x251* es idéntico al '151 en sus terminales de salida y en su diseño de lógica interna, a excepción que Y y Y_L son salidas de tres estados. Cuando la entrada EN_L es negada, en lugar de forzar las salidas para que sean negadas, éstas son forzadas al estado Hi-Z. De manera similar, el *74x253* y el *74x257* son versiones de tres estados del '153 y del '157. Las salidas de tres estados son especialmente útiles cuando los multiplexores de n entradas se combinan para formar multiplexores más grandes, como se sugiere en la siguiente subsección.

5.7.2 Expansión de multiplexores

Rara vez el tamaño de un multiplexor MSI satisface las características del problema práctico. Por ejemplo, sugerimos al principio que un multiplexor de 16 bits y 8 entradas podría ser utilizado en el diseño de un procesador de computadora. Esta función podría ser realizada mediante 16 multiplexores 74x151 de 8 entradas, 1 bit o celdas equivalentes ASIC, si cada una manejara un bit de todas las entradas y la salida. El campo de selección de registro de 3 bits del procesador estaría conectado a las entradas A, B y C de los 16 multiplexores, de modo que todos seleccionarían la misma fuente de registro en cualquier momento dado.

El dispositivo que produce el campo de selección de registro de 3 bits en este ejemplo debe tener suficiente fan out para manejar 16 cargas. Con los CI de la serie 74LS, esto es posible porque los dispositivos típicos tienen un fan out de 20 cargas LS-TTL.

Con todo, es afortunado que el '151 fuera diseñado de tal modo que las entradas A, B y C presenten solamente una carga LS-TTL al circuito que lo controla. Teóricamente, el '151 podría haber sido diseñado sin el primer conjunto de tres inversores mostrado en las entradas de selección en la figura 5-63, pero entonces cada entrada de selección habría presentado cinco cargas LS-TTL y los controladores en la aplicación de selección de registro necesitarían un fan out de 80.

Otra dimensión en la que los multiplexores pueden expandirse es el número de fuentes de datos. Por ejemplo, supongamos que necesitamos un multiplexor de 32 entradas y 1 bit. La figura 5-66 ilustra una manera de construirlo. Se requieren cinco bits de selección. Un decodificador de 2 a 4 (una mitad del 74x139) decodifica los dos bits de selección de mayor orden para habilitar uno de cuatro multiplexores 74x151 de 8 entradas. Puesto que solamente un '151 se encuentra habilitado a la vez, las salidas del '151 pueden funcionar con un dispositivo OR para obtener la salida final.

FAN OUT DE LA SEÑAL DE CONTROL EN ASICS

La clase de consideración sobre fan out que describimos con anterioridad ocurre con mucha frecuencia en el diseño ASIC. Cuando un conjunto de señales de control tales como el campo de selección de registro en el ejemplo, controla un gran número de bits, el fan out que se requiere puede ser enorme. En chips CMOS, la consideración no es la carga de CD, sino la carga capacitiva, la cual disminuye el rendimiento. En una aplicación así, el diseñador debe dividir cuidadosamente la carga y seleccionar los puntos donde se deben separar las señales de control para reducir el fan out. Mientras se insertan los buffers adicionales, el diseñador debe tener cuidado de no aumentar en forma considerable el área del chip, y debe evitar conectar demasiados buffers en serie porque el retardo crecería hasta un valor inaceptable.

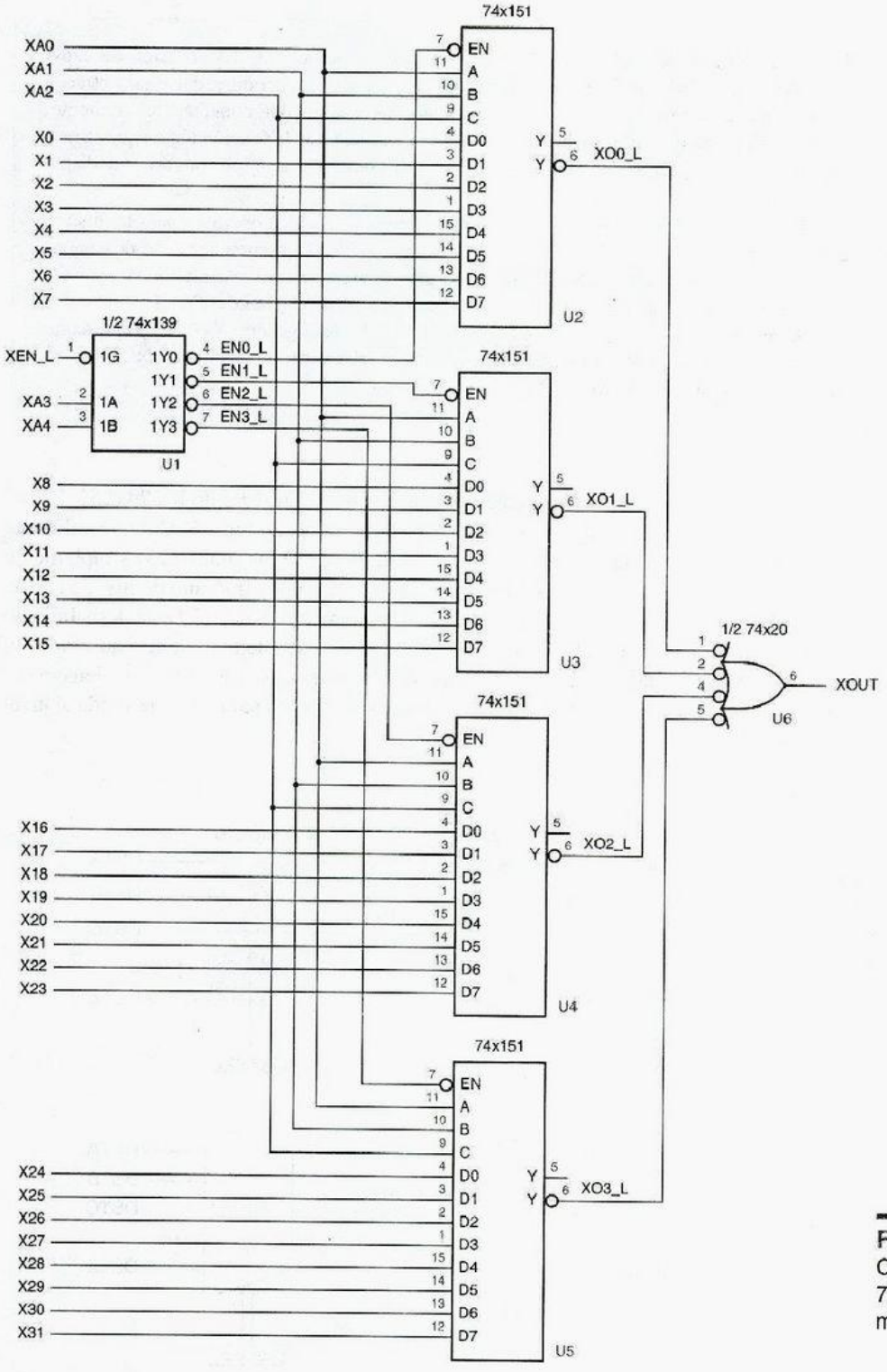


Figura 5-66
Combinación de varios 74x151 para hacer un multiplexor de 32 a 1.

ENCENDIDO DE LA MÁQUINA DE BURBUJAS

El uso del diseño de lógica burbuja-burbuja debería mejorar la comprensión de estos ejemplos de diseño de multiplexor. Puesto que las salidas del decodificador y las entradas de habilitación del multiplexor son todas activas bajas se pueden conectar directamente. Se pueden ignorar las burbujas de inversión cuando se piensa en la función lógica que se realiza: cuando una salida de decodificador particular es asertiva, el multiplexor correspondiente es habilitado.

El diseño de lógica burbuja-burbuja también proporciona dos opciones para la función OR final en la figura 5-66. El diseño más obvio habría utilizado una compuerta OR de 4 entradas conectada a las salidas Y. Sin embargo, para un funcionamiento más rápido, empleamos una compuerta invertida, uña NAND de 4 entradas conectada a las salidas /Y. Esto elimina el retardo de dos inversores: el que se utiliza dentro del '151 para generar Y de /Y, y el circuito inversor extra que se utiliza para obtener una función OR a partir de un circuito NOR básico en una compuerta OR CMOS o TTL.

El multiplexor de 32 a 1 también puede construirse utilizando los 74x251. El circuito es idéntico al de la figura 5-66, excepto que la compuerta NAND de salida es eliminada. En cambio, las salidas Y (y si se desean, Y_L) de los cuatro '251 simplemente se unen entre sí. El decodificador '139 asegura que como máximo uno de los '251 tiene sus salidas de tres estados habilitadas en cualquier momento. Si el '139 está deshabilitado (XEN_L se encuentra negada), entonces todos los '251 se encuentran deshabilitados, y las salidas XOUT y XOUT_L están indefinidas. Sin embargo, si se desea, pueden conectarse resistores desde cada una de estas señales hasta +5 volts para llevar la salida al nivel ALTO en este caso.

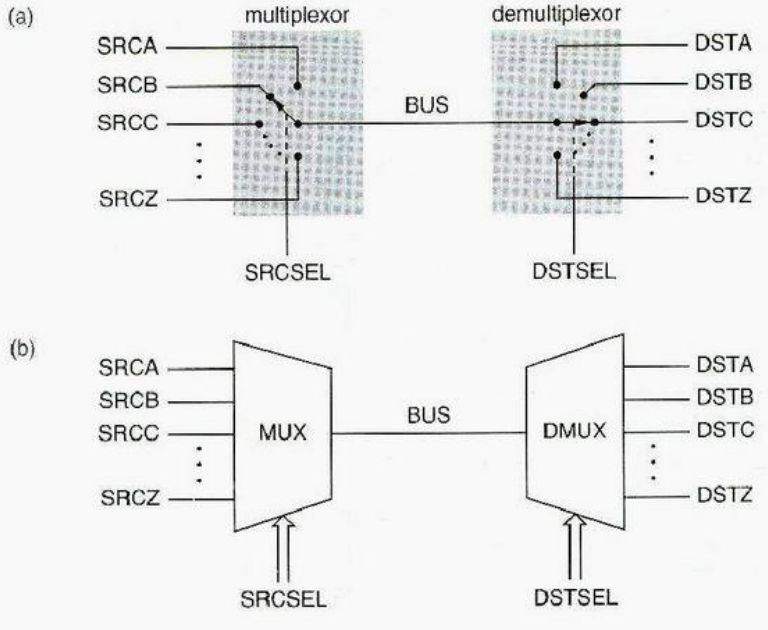


Figura 5-67
Un multiplexor que controla un bus y un demultiplexor que recibe el bus: a) equivalente de interruptor; b) símbolos de diagrama de bloques.

5.7.3 Multiplexores, demultiplexores y buses

Un multiplexor puede ser utilizado para seleccionar una de n fuentes de datos para transmitirse sobre un bus. En el otro extremo del bus, puede utilizarse un *demultiplexor* para dirigir los datos del bus a uno de m destinos. Una aplicación así, utilizando un bus de 1 bit, es descrita en términos de nuestra analogía del interruptor en la figura 5-67(a). De hecho, los diagramas de bloques para circuitos lógicos a menudo describen multiplexores y demultiplexores utilizando los símbolos con forma de cuña en (b), para sugerir de manera visual cómo una selección de entre múltiples fuentes de datos se consigue direccionar sobre un bus y dirigirse a una selección de entre múltiples destinos.

demultiplexor

La función de un demultiplexor es justamente la inversa de la de un multiplexor. Por ejemplo, un demultiplexor de n salidas y 1 bit tiene una entrada de datos y s entradas para seleccionar una de $n = 2^s$ salidas de datos. En operación normal, todas las salidas excepto la seleccionada son 0; la salida seleccionada es igual a la entrada de datos. Esta definición puede generalizarse para un demultiplexor de n salidas y b bits; un dispositivo así tendría b entradas de datos y sus s entradas de selección eligen una de $n = 2^s$ conjuntos de b salidas de datos.

Un decodificador binario con una entrada de habilitación puede utilizarse como un demultiplexor, como se muestra en la figura 5-68. La entrada de habilitación del decodificador se conecta a la línea de datos y sus entradas de selección determinan cuál de sus líneas de salida es controlada con el bit de datos. Las líneas de salida restantes son negadas. De este modo, el 74x139 puede utilizarse como un demultiplexor de 2 bits y 4 salidas con entradas y salidas de datos activas bajas y el 74x138 puede ser empleado como un demultiplexor de 1 bit y 8 salidas. De hecho, el catálogo del fabricante presenta por lo regular estos CI como "decodificadores/demultiplexores".

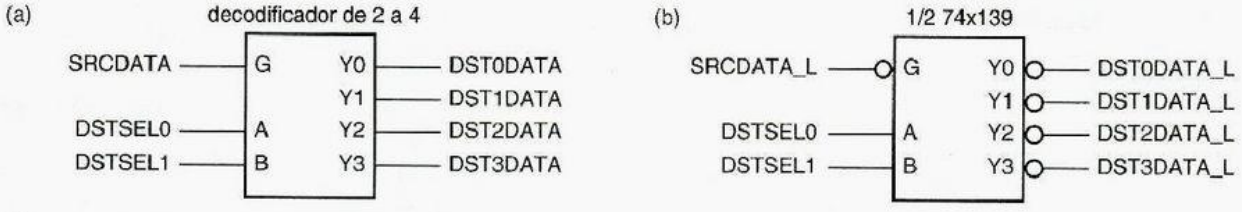


Figura 5-68 Uso de un decodificador binario de 2 a 4 como un demultiplexor de 4 salidas y 1 bit: a) decodificador genérico; b) 74x139.

5.7.4 Multiplexores en ABEL y PLD

Los multiplexores son muy fáciles de diseñar empleando ABEL y PLD combinacionales. Por ejemplo, la función de un multiplexor 74x153 de 2 bits y 4 entradas puede duplicarse en un PAL16L8 como se ilustra en la figura 5-69 y la tabla 5-37. Vale la pena fijarse en varias características del diseño y programación basados en PLD:

- Los nombres de señal en el programa ABEL varían ligeramente de los nombres de señal para un 74x153 en la figura 5-65 de la página 401, puesto que ABEL no permite que se utilice un número como el primer carácter de un nombre de señal.
- Un 74x153 tiene doce entradas, mientras que un PAL16L8 tiene solamente diez entradas. Por lo tanto, dos de las entradas del '153 son asignadas a las terminales de E/S del 16L8, las cuales ya no son utilizables como salidas.

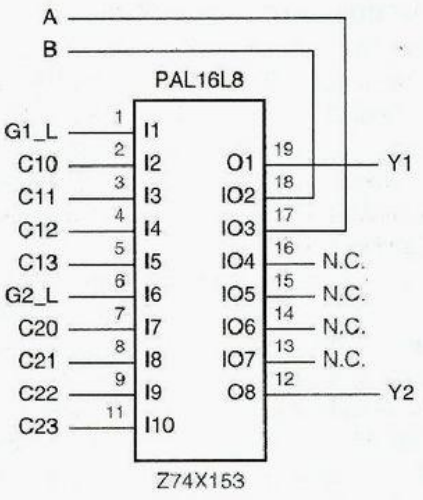


Figura 5-69
Diagrama lógico para el PAL16L8 usado como un multiplexor tipo 74x153.

- Las salidas del '153 (1Y y 2Y) se asignan a las terminales 19 y 12 del 16L8, las cuales son utilizables *solamente* como salidas. Esto se prefiere sobre la asignación a las terminales de E/S; dada una elección, es mejor dejar terminales de E/S que terminales sólo de salida como repuestos.
- Aunque las ecuaciones del multiplexor en la tabla están escritas de manera bastante natural en la forma de suma de productos, no se mapean directamente sobre la

Tabla 5-37
Programa ABEL para un multiplexor de 2 bits y 4 entradas tipo 74x153.

```
module Z74X153
title '74x153-like multiplexer PLD '
Z74X153 device 'P16L8';

" Input and output pins
A, B, G1_L, G2_L          pin 17, 18, 1, 6;
C10, C11, C12, C13        pin 2, 3, 4, 5;
C20, C21, C22, C23        pin 7, 8, 9, 11;
Y1, Y2                    pin 19, 12 istype 'com';

" Active-level conversion
G1 = !G1_L;  G2 = !G2_L;

equations
Y1 = G1 & ( !B & !A & C10
          # !B &  A & C11
          #  B & !A & C12
          #  B &  A & C13);

Y2 = G2 & ( !B & !A & C20
          # !B &  A & C21
          #  B & !A & C22
          #  B &  A & C23);
end Z74X153
```

```
!Y1 = (!B & !A & !C10
     # !B &  A & !C11
     #  B & !A & !C12
     #  B &  A & !C13
     #  G1);
!Y2 = (!B & !A & !C20
     # !B &  A & !C21
     #  B & !A & !C22
     #  B &  A & !C23
     #  G2);
```

Tabla 5-38
Ecuaciones reducidas e invertidas para el multiplexor de 2 bits y 4 entradas tipo 74x153.

estructura del 16L8, debido al inversor entre el arreglo AND-OR y las terminales de salida reales. Por consiguiente, el compilador ABEL debe complementar las ecuaciones en la tabla y, entonces, reducir el resultado a la forma de suma de productos. Con un GAL16V8, podría utilizarse cualquier versión de las ecuaciones.

Las funciones del multiplexor son todavía más fáciles de expresar utilizando conjuntos y relaciones de ABEL. Por ejemplo, la tabla 5-39 muestra el programa ABEL para un multiplexor de 8 bits y 4 entradas. No se incluye la instrucción `device`, porque esta función tiene demasiadas entradas y salidas que adaptar en cualquiera de los PLD que hemos descrito hasta ahora. Sin embargo, es bastante obvio que un multiplexor de cualquier tamaño puede especificarse en apenas unas cuantas líneas de código de este modo.

```
module mux4in8b
title '4-input, 8-bit wide multiplexer PLD'

" Input and output pins
!G                pin;              " Output enable for Y bus
S1..S0            pin;              " Select inputs, 0-3 ==> A-D
A1..A8, B1..B8, C1..C8, D1..D8 pin; " 8-bit input buses A, B, C, D
Y1..Y8            pin istype 'com'; " 8-bit three-state output bus

" Sets
SEL = [S1..S0];
A = [A1..A8];
B = [B1..B8];
C = [C1..C8];
D = [D1..D8];
Y = [Y1..Y8];

equations
Y.OE = G;
WHEN (SEL == 0) THEN Y = A;
ELSE WHEN (SEL == 1) THEN Y = B;
ELSE WHEN (SEL == 2) THEN Y = C;
ELSE WHEN (SEL == 3) THEN Y = D;
end mux4in8b
```

Tabla 5-39
Programa ABEL para un multiplexor de 8 bits y 4 entradas.

Tabla 5-40
Tabla de funciones para un multiplexor especializado de 18 bits y 4 entradas.

S2	S1	S0	*Entrada para selección*
0	0	0	A
0	0	1	B
0	1	0	A
0	1	1	C
1	0	0	A
1	0	1	D
1	1	0	A
1	1	1	B

Del mismo modo, es fácil personalizar funciones del multiplexor con la ayuda de ABEL. Por ejemplo, supongamos que se necesita un circuito que seleccione uno de cuatro buses de 18 bits, A, B, C o D, para controlar un bus F de salida de 18 bits, como se especificó en la tabla 5-40 mediante tres bits de control. Existen más combinaciones de bits de control que entradas de multiplexor, así que un multiplexor estándar de 4 entradas no es suficiente para cumplir los requisitos (véase el ejercicio 5-61). Puede diseñarse un multiplexor de 4 entradas y 3 bits con el comportamiento requerido para adaptarse en un PAL16L8 o GAL16V8 como se ilustra en la figura 5-70 y la tabla 5-41, y pueden emplearse seis copias de este dispositivo para hacer el multiplexor de 18 bits. De manera alternativa, podría emplearse un solo PLD más grande. En cualquiera de los casos, el programa ABEL es muy fácil de modificar para diferentes criterios de selección.

Puesto que esta función utiliza todas las terminales disponibles en el PAL16L8, tendríamos que hacer la asignación de terminales en la figura 5-70 cuidadosamente. En particular, tendríamos que asignar dos señales de salida a las dos terminales sólo de salida (O1 y O8) a fin de maximizar el número de terminales de entrada disponibles.

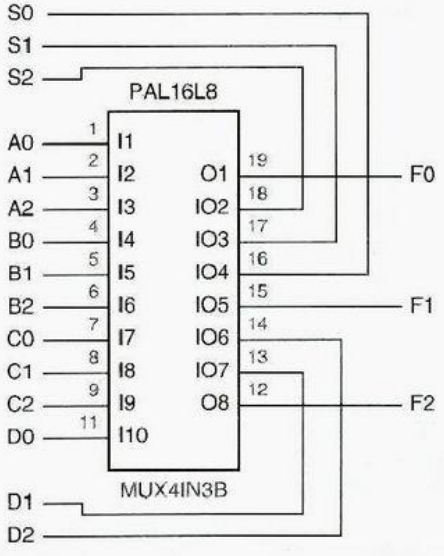


Figura 5-70
Diagrama lógico para el PAL16L8 que se utiliza como un multiplexor especializado de 3 bits y 4 entradas.

Tabla 5-41 Programa ABEL para un multiplexor especializado de 3 bits y 4 entradas.

```
module mux4in3b
title 'Specialized 4-input, 3-bit Multiplexer'
mux4in3b device 'P16L8';

" Input and output pins
S2..S0                          pin 16..18;                  " Select inputs
A0..A2, B0..B2, C0..C2, D0..D2  pin 1..9, 11, 13, 14;        " Bus inputs
F0..F2                          pin 19, 15, 12 istype 'com'; " Bus outputs

" Sets
SEL = [S2..S0];
A = [A0..A2];
B = [B0..B2];
C = [C0..C2];
D = [D0..D2];
F = [F0..F2];

equations
WHEN (SEL== 0) # (SEL== 2) # (SEL== 4) # (SEL== 6) THEN F = A;
ELSE WHEN (SEL== 1) # (SEL== 7) THEN F = B;
ELSE WHEN (SEL== 3) THEN F = C;
ELSE WHEN (SEL== 5) THEN F = D;

end mux4in3b
```

MÁS FÁCIL, PERO NO MÁS BARATO

Como usted ha visto, es muy fácil programar un PLD para efectuar funciones de decodificador y multiplexor. No obstante, si se necesita la función lógica de un decodificador o multiplexor estándar, por lo regular es menos costoso utilizar un chip estándar MSI que emplear un PLD. El enfoque basado en PLD es mejor si el multiplexor tiene algunos requerimientos funcionales no estándar, o si usted piensa que puede tener que cambiar su función como resultado de la depuración.

5.7.5 Multiplexores en VHDL

Los multiplexores son muy fáciles de describir en VHDL. En el estilo de flujo de datos de arquitectura, la instrucción `SELECT` proporciona la funcionalidad requerida, como se muestra en la tabla 5-42, la descripción VHDL del multiplexor de 4 entradas y 8 bits.

En una arquitectura de comportamiento, se emplea una instrucción `CASE`. Por ejemplo, la tabla 5-43 muestra una arquitectura basada en procesos para la misma entidad `mux4in8b`.

Como en ABEL, es muy fácil personalizar el criterio de selección en un programa multiplexor VHDL. Por ejemplo, la tabla 5-44 es un programa en estilo de comportamiento para un multiplexor especializado de 4 entradas y 18 bits con el criterio de selección de la tabla 5-40.

En cada ejemplo, si las entradas seleccionadas no son válidas (por ejemplo que contengan `U` o `X`), el bus de salida se establece como "desconocido" para ayudar a capturar errores durante la simulación.

Tabla 5-42 Programa VHDL de flujo de datos para un multiplexor de 8 bits y 4 entradas.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4in8b is
    port (
        S: in STD_LOGIC_VECTOR (1 downto 0);          -- Select inputs  0-3 ==> A-D
        A, B, C, D: in STD_LOGIC_VECTOR (1 to 8);     -- Data bus input
        Y: out STD_LOGIC_VECTOR (1 to 8)              -- Data bus output
    );
end mux4in8b;

architecture mux4in8b of mux4in8b is
begin
    with S select Y <=
      A when "00".
      B when "01",
      C when "10",
      D when "11",
      (others => 'U') when others; -- this creates an 8 bits vector of 'U'
end mux4in8b;
```

Tabla 5-43 Arquitectura de comportamiento para un multiplexor de 8 bits y 4 entradas.

```
architecture mux4in8p of mux4in8b is
begin
process(S, A, B, C, D)
  begin
    case S is
      when "00" => Y <= A;
      when "01" => Y <= B;
      when "10" => Y <= C;
      when "11" => Y <= D;
      when others => Y <= (others => 'U'); -- 8-bit vector of 'U'
    end case;
  end process;
end mux4in8p;
```

5.8 Compuertas OR Exclusivas y circuitos de paridad

5.8.1 Compuertas OR Exclusivas y NOR Exclusivas

OR Exclusivo (XOR)
NOR Exclusivo (XNOR)
Equivalencia

Una compuerta *OR Exclusivo (XOR)* es una compuerta de 2 entradas cuya salida es 1 si exactamente una de sus entradas es 1. Visto de otra manera, una compuerta XOR produce un 1 si sus entradas son diferentes. Una compuerta *NOR Exclusivo (XNOR)* o *de Equivalencia* es precisamente la opuesta: produce un 1 si sus entradas son las mismas. Una

Tabla 5-44 Programa VHDL de comportamiento para un multiplexor de 3 bits y 4 entradas.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4in8b is
    port (
       S: in STD_LOGIC_VECTOR (2 dowto 0);              -- Select inputs
                                                           0-7 ==> ABACADAS
       A, B, C, D: in STD_LOGIC_VECTOR (1 to 18);       -- Data bus inputs
       Y: out STD_LOGIC_VECTOR (1 to 18)                -- Data bus output
    );
end mux4in3b;

architecture mux4in3b of mux4in3b is
begin
process (S, A, B, C, D)
variable i: INTEGER;
  begin
    case S is
      when "000" | "010" | "100" | "110" => Y <= A;
      when "001" | "111" => Y <= B;
      when "011" => Y <= C;
      when "101" => Y <= D;
      when others => Y <= (others => 'U'); -- 18-bit vector of 'U'
    end case;
  end process;
end mux4in3p;
```

tabla de verdad para estas funciones se muestra en la tabla 5-45. La operación XOR en ocasiones se denota mediante el símbolo "⊕", es decir,

$$X \oplus Y = X' \cdot Y + X \cdot Y'$$

Aunque el OR EXCLUSIVO no es una de las funciones básicas del álgebra de conmutación, las compuertas XOR discretas son bastante utilizadas comúnmente en la práctica. La mayoría de las tecnologías de conmutación no pueden efectuar la función XOR en forma directa; en cambio, emplean diseños de compuertas múltiples como los mostrados en la figura 5-71.

X	*Y*	*X⊕Y (XOR)*	*(X⊕Y)′ (XNOR)*
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 5-45
Tabla de verdad para funciones XOR y XNOR.

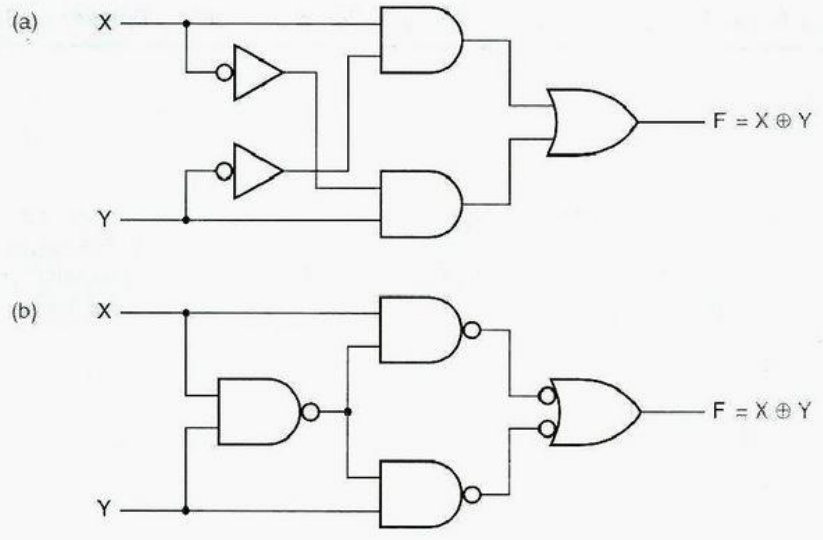


Figura 5-71
Diseños de compuerta múltiple para la función XOR de 2 entradas: a) AND-OR; b) NAND de tres niveles.

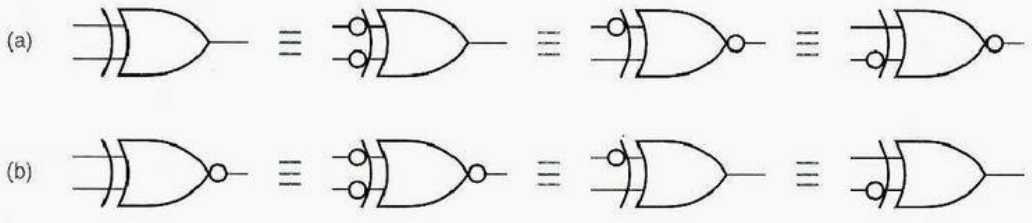


Figura 5-72 Símbolos equivalentes para a) compuertas XOR; b) compuertas XNOR.

Los símbolos lógicos para las funciones XOR y XNOR se muestran en la figura 5-72. Existen cuatro símbolos equivalentes para cada función. Todas estas alternativas son consecuencia de una regla simple:

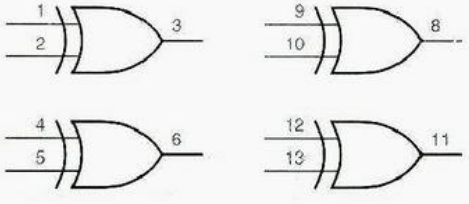
- Cualesquiera dos señales (entradas o salidas) de una compuerta XOR o XNOR pueden complementarse sin cambiar la función lógica resultante.

En el diseño lógico en burbuja, seleccionamos el símbolo que expresa mejor la función lógica que se está realizando.

74x86

Cuatro compuertas XOR se proporcionan en un solo CI SSI de 14 terminales, el *74x86* ilustrado en la figura 5-73. Las nuevas familias lógicas SSI no ofrecen compuertas XNOR, aunque están fácilmente disponibles en las bibliotecas ASIC y FPGA y como primitivas en los HDL.

Figura 5-73
Terminales externas de la compuerta XOR de 2 entradas 74x86 cuádruple.



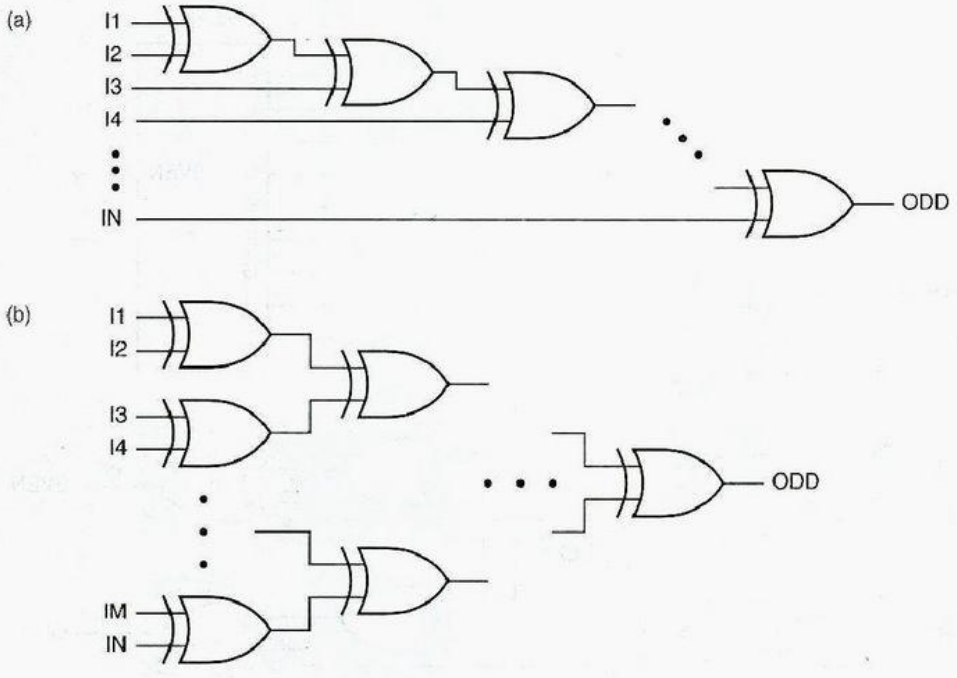


Figura 5-74
Compuertas XOR en cascada: a) conexión en cadena; b) estructura de árbol.

5.8.2 Circuitos de paridad

Como se muestra en la figura 5-74(a), *n* compuertas XOR pueden conectarse en cascada para formar un circuito con $n + 1$ entradas y una sola salida. Esto se conoce como un *circuito de paridad impar*, porque su salida es 1, si un número impar de sus entradas es 1. El circuito en (b) también es uno de paridad impar, pero es más rápido porque sus compuertas se conectan en una estructura tipo árbol. Si la salida de cualquier circuito es invertida, obtenemos un *circuito de paridad par*, cuya salida es 1 si un número par de sus entradas es 1.

circuito de paridad impar

circuito de paridad par

5.8.3 El generador de paridad de 9 bits 74x280

En lugar de construir un circuito de paridad de bits múltiples con compuertas XOR discretas, es más económico poner todas las compuertas XOR en un solo encapsulado MSI, que solamente tiene disponibles las entradas y salidas primarias en las terminales externas. El generador de paridad de 9 bits *74x280*, que se muestra en la figura 5-75, es un dispositivo de esta clase. Tiene nueve entradas y dos salidas que indican si un número par o impar de entradas es 1.

74x280

5.8.4 Aplicaciones de verificación de paridad

En la sección 2.15 describimos los códigos de detección de errores que hacen uso de un bit extra, denominado el bit de paridad, para detectar errores en la transmisión y almacenamiento de los datos. En un código de paridad par, el bit de paridad se elige de modo que el número total de bits 1 en una palabra de código sea par. Los circuitos de paridad como el 74x280 se utilizan tanto para generar el valor correcto del bit de paridad, cuando una palabra de código es almacenada o transmitida, como para verificar el bit de paridad, cuando una palabra de código es recuperada o recibida.

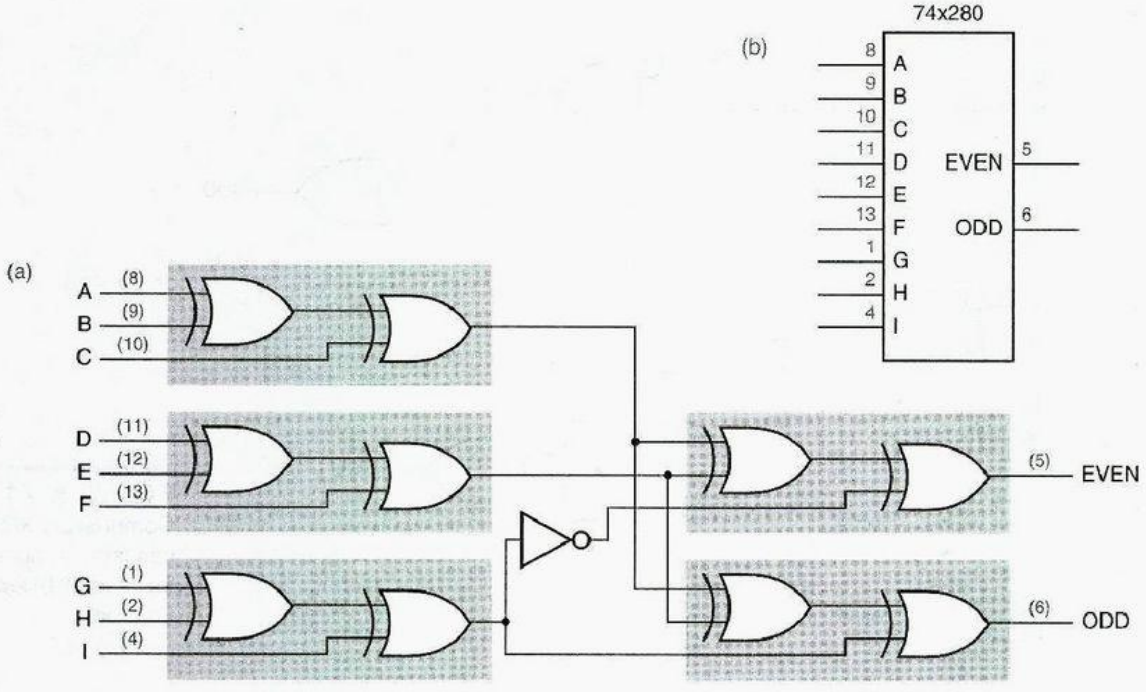


Figura 5-75 El generador de paridad impar/par de 9 bits 74x280: a) diagrama lógico, incluyendo los números de terminales para un encapsulado en doble línea de 16 bits estándar; b) símbolo lógico tradicional.

La figura 5-76 muestra cómo se puede utilizar un circuito de paridad para detectar errores en la memoria de un sistema de microprocesador. La memoria almacena bytes de 8 bits, más un bit de paridad para cada byte. El microprocesador utiliza un bus bidireccional D[0:7] para transferir los datos hacia y desde la memoria. Se emplean dos líneas de control RD y WR, para indicar si se desea una operación de lectura o escritura, y una señal de ERROR se activa para indicar errores de paridad durante las operaciones de lectura. No se muestran detalles completos de los chips de memoria, tales como las entradas de direccionamiento; los chips de memoria se describen con detalle en el capítulo 10. Para verificar la paridad, solamente nos interesan las conexiones de datos a la memoria.

ACELERACIÓN DEL ÁRBOL XOR

Si cada compuerta XOR de la figura 5-75 se construyera utilizando compuertas discretas NAND como en la figura 5-71(b), el 74x280 sería bastante lento, tendría un retardo de propagación equivalente a $4 \cdot 3 + 1$, o 13, compuertas NAND. En cambio, en una implementación típica del 74x280 que utiliza una compuerta AND-OR-INVERT para realizar la función de cada par sombreado de compuertas XOR de la figura, y ofrece aproximadamente el mismo retardo que una compuerta NAND simple. Las entradas A–I se regulan a través de dos niveles de inversores, de modo que cada entrada presente sólo una carga unitaria al circuito que la controla. De este modo, el retardo de propagación total a través de esta implementación del 74x280 es aproximadamente el mismo que el de cinco compuertas inversoras, no 13.

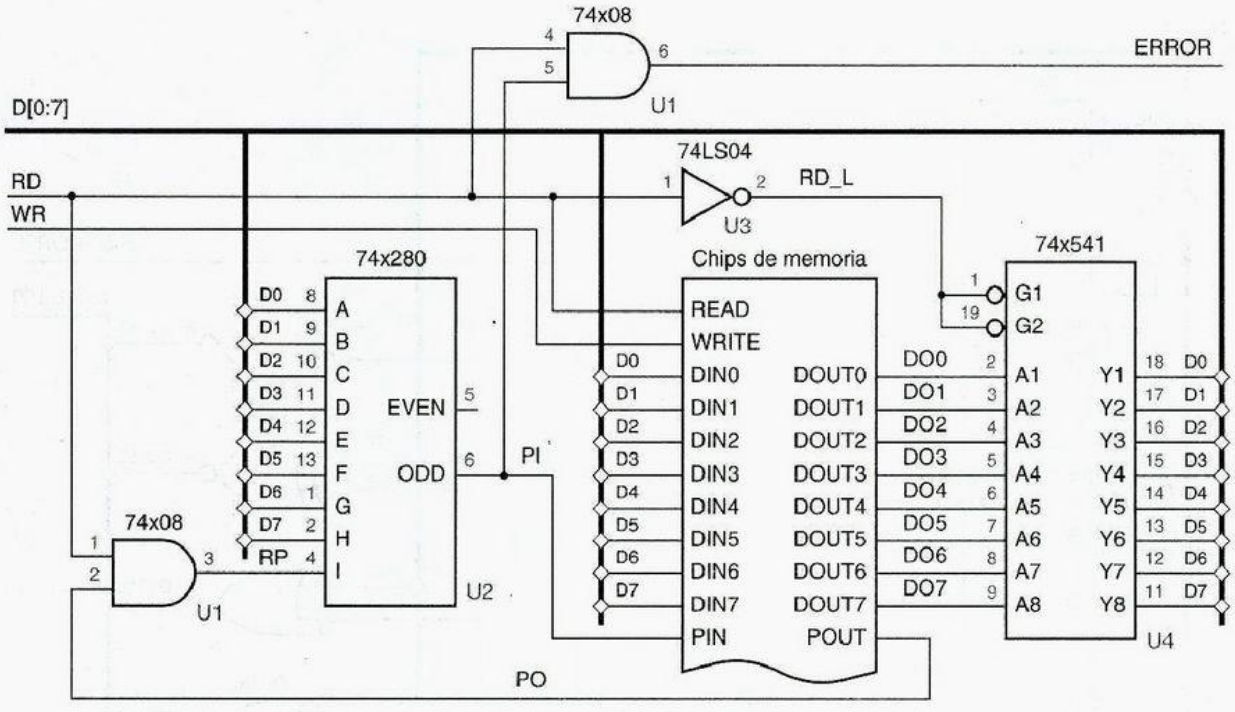


Figura 5-76 Generación de paridad y verificación para un sistema de memoria de 8 bits de ancho.

Para almacenar un byte en los chips de memoria, especificamos una dirección (que no se muestra), colocamos el byte en D[0–7], generamos su bit de paridad en PIN y activamos WR. La compuerta AND en la entrada I del 74x280 asegura que I es 0, excepto durante las operaciones de lectura, de modo que durante las escrituras la salida del '280 depende solamente de la paridad de los datos del bus D. La salida ODD del '280 está conectada a PIN, de tal forma que el número total de unos almacenados es par.

Para recuperar un byte, especificamos una dirección (que no se muestra) y activamos RD; el valor del byte aparece en DOUT[0–7] y su paridad se presenta en POUT. Un 74x541 controla el byte en el bus D, y el '280 verifica su paridad. Si la paridad de la palabra de 9 bits DOUT[0–7],POUT es impar durante una lectura, la señal de ERROR se afirma.

Los circuitos de paridad también se utilizan con códigos de corrección de errores como los códigos de Hamming, que se describen en la sección 2.15.3. Mostramos la matriz de verificación de paridad para un código Hamming de 7 bits en la figura 2-13 de la página 63. Podemos corregir errores en este código como se muestra en la figura 5-77. Una palabra de 7 bits, posiblemente con un error, se presenta en DU[1–7]. Tres 74x280 calculan la paridad de los tres grupos de bits definidos por la matriz de verificación de paridad. Las salidas de los '280 forman el síndrome, el cual es el número del bit de entrada erróneo, si existe. Se emplea un 74x138 para decodificar el síndrome. Si éste es cero, la señal NOERROR_L es asertiva (la cual también podría llamarse ERROR). De otro

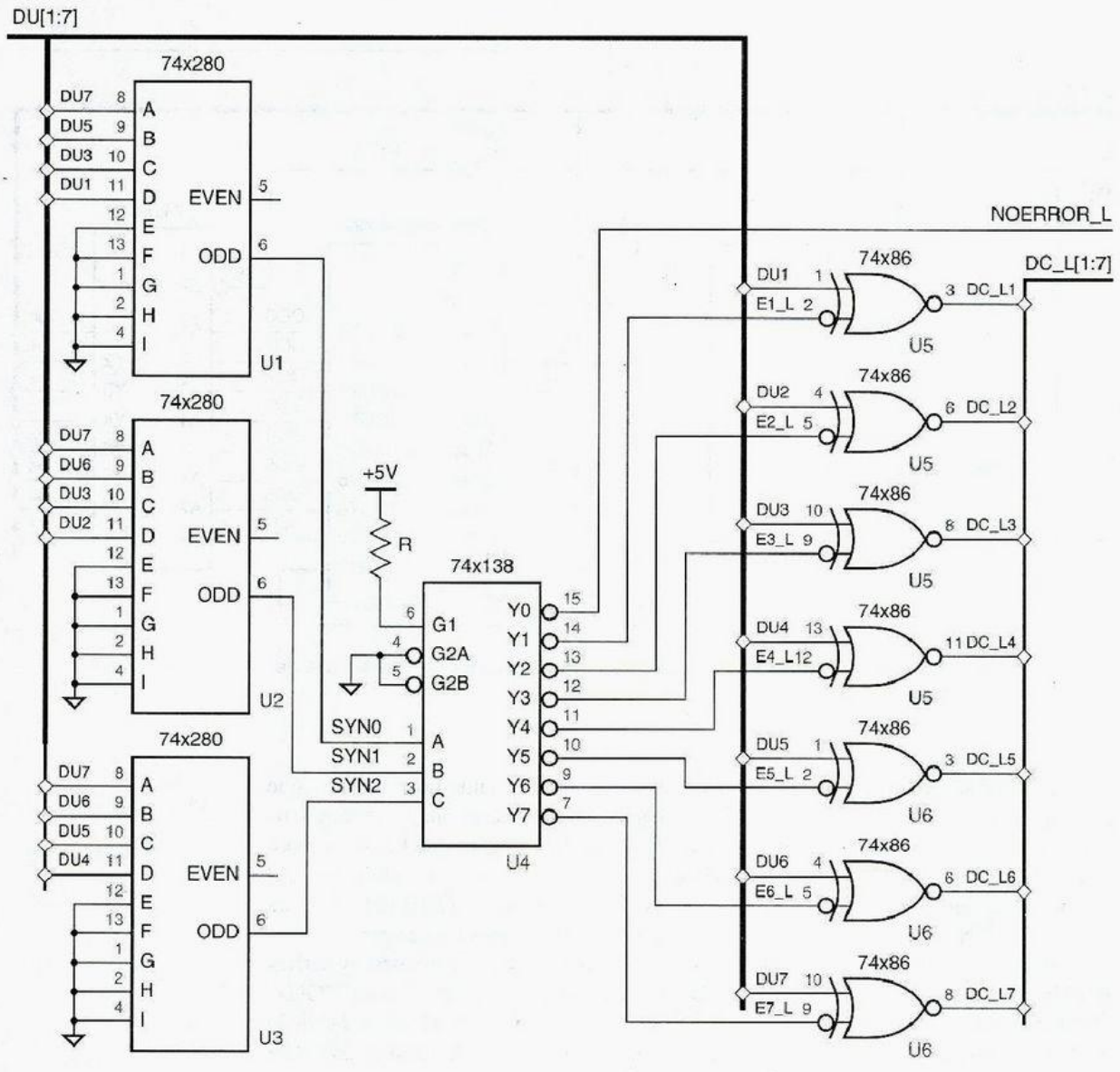


Figura 5-77 Circuito de corrección de errores para un código Hamming de 7 bits.

modo, el bit erróneo se corrige mediante complementación. La palabra de código corregida aparece en el bus DC_L.

Advierta que las salidas activas bajas del '138 nos llevan a utilizar un bus activo bajo DC_L. Si requerimos un bus DO activo alto, podríamos haber puesto un inversor discreto en cada entrada o salida de XOR, o haber empleado un decodificador con salidas activas bajas, o haber usado compuertas XNOR.

5.8.5 Compuertas OR Exclusiva y circuitos de paridad en ABEL y PLD

La función OR Exclusivo es denotada en ABEL por el operador $ y su complemento, la función NOR Exclusivo, se denota por !$. En principio, estos operadores pueden utilizarse libremente en expresiones ABEL. Por ejemplo, se podría especificar una salida PLD equivalente a la salida EVEN del 74x280 utilizando la siguiente ecuación ABEL:

```
EVEN = !(A $ B $ C $ D $ E $ F $ G $ H $ I);
```

Sin embargo, la mayoría de los PLD realizan expresiones mediante el uso de lógica AND-OR de dos niveles y tienen poca, si es que alguna, capacidad de realización de funciones XOR directamente. Por desgracia, el mapa de Karnaugh de una función XOR de n entradas es un tablero de ajedrez con 2^{n-1} implicantes principales. De este modo, la realización del producto de sumas de la ecuación simple anterior requiere de 256 términos de producto, mucho más allá de la capacidad de cualquier PLD.

Los denominados PLD serie X realizan una función XOR de dos entradas directamente en una estructura de tres niveles que utiliza una compuerta XOR simple para combinar dos productos independientes de sumas. Esta estructura resulta ser útil en el diseño de contadores. Sin embargo, para crear funciones XOR mayores, un diseñador de tablero normalmente debe utilizar un componente especializado generador/verificador de paridad, como el 74x280, y un diseñador ASIC debe combinar compuertas XOR individuales en un árbol de paridad de múltiples niveles, similar al de la figura 5-74(b) de la página 413.

5.8.6 Compuertas OR Exclusivo y circuitos de paridad en VHDL

Al igual que ABEL, VHDL proporciona operadores primitivos, `xor` y `xnor`, para especificar funciones XOR y XNOR (`xnor` está disponible solamente en VHDL-93). Por ejemplo, la tabla 5-46 es un programa estilo flujo de datos para un dispositivo XOR de 3 entradas que utiliza la primitiva `xor`. También es posible especificar XOR o funciones de paridad en forma de comportamiento, como se hace en la tabla 5-47 para una función de paridad de 9 entradas semejante a la 74x280.

Tabla 5-46 Programa VHDL estilo de flujo de datos para un dispositivo XOR de 3 entradas.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity vxor3 is
    port (
        A, B, C: in STD_LOGIC;
        Y: out STD_LOGIC
    );
end vxor3;

architecture vxor3 of vxor3 is
begin
  Y <= A xor B xor C;
end vxor3;
```

Tabla 5-47 Programa VHDL de comportamiento para un verificador de paridad de 9 entradas.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity parity9 is
    port (
        I: in STD_LOGIC_VECTOR (1 to 9);
        EVEN, ODD: out STD_LOGIC
    );
end parity9;

architecture parity9p of parity9 is
begin
process (I)
  variable p : STD_LOGIC;
  begin
    p := I(1);
    for j in 2 to 9 loop
      if I(j) = '1' then p := not p; end if;
    end loop;
    ODD <= p;
    EVEN <= not p;
  end process;
end parity9p;
```

Cuando un programa VHDL que contiene funciones XOR grandes es sintetizado, la herramienta de síntesis lo hará mejor si puede realizar la función en la tecnología del dispositivo objetivo. No es cuestión de magia, si intentamos implantar el programa VHDL de la tabla 5-47 para un PLD 16V8, ¡no cabrá!

Las bibliotecas típicas ASIC y FPGA contienen funciones XNOR y XOR de dos y tres entradas como primitivas. En los ASIC y CMOS, estas entradas primitivas por lo regular se realizan en forma eficiente al nivel de transistores empleando compuertas de transmisión, como se muestra en los ejercicios 5.56 y 5.73. Pueden construirse árboles XOR rápidos y compactos utilizando estas primitivas. Sin embargo, las herramientas de síntesis típicas VHDL no son lo suficientemente inteligentes para crear una estructura de árbol eficaz desde un programa de comportamiento como el de la tabla 5-47. En su lugar, podemos utilizar un programa estructural para obtener exactamente lo que queremos.

Por ejemplo, la tabla 5-48 es un programa estructural para una función XOR de 9 entradas que es equivalente al 74x280 de la figura 5-75(a) en estructura y función. En este ejemplo, hemos empleado el componente previamente definido `vxor3` como el bloque de construcción básico del árbol XOR. En un ASIC, reemplazaríamos el `vxor3` con una primitiva XOR de 3 entradas de la biblioteca ASIC. También, si se tuviera disponible un XNOR de 3 entradas, podríamos eliminar la inversión explícita para `Y3N` y, en su lugar utilizar, el XNOR para `U5`, empleando la señal Y3 no invertida como su última entrada.

Nuestro ejemplo final es una versión VHDL del circuito decodificador de Hamming de la figura 5-77 y que se muestra en la tabla 5-49. Se define una función `syndrome (DU)`

Tabla 5-48 Programa VHDL estructural para un verificador de paridad tipo 74x280.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V74x280 is
    port (
        I: in STD_LOGIC_VECTOR (1 to 9);
        EVEN, ODD: out STD_LOGIC
    );
end V74x280;

architecture V74x280s of V74x280 is
component vxor3
 port (A, B, C: in STD_LOGIC; Y: out STD_LOGIC);
end component;
signal Y1, Y2, Y3, Y3N: STD_LOGIC;
begin
  U1: vxor3 port map (I(1), I(2), I(3), Y1);
  U2: vxor3 port map (I(4), I(5), I(6), Y2);
  U3: vxor3 port map (I(7), I(8), I(9), Y3);
  Y3N <= not Y3;
  U4: vxor3 port map (Y1, Y2, Y3, ODD);
  U5: vxor3 port map (Y1, Y2, Y3N, EVEN);
end V74x280s;
```

para devolver el síndrome de 3 bits de un vector de entrada DU de datos de 7 bits sin corregir. En el proceso "principal", el vector de entrada DC de datos corregidos se establece, al inicio, igual a DU. La función CONV_INTEGER, que definimos en la sección 4.7.4, se utiliza para convertir el síndrome de 3 bits a un entero. Si el síndrome es distinto de cero, el bit correspondiente de DC se complementa para corregir el supuesto error de 1 bit. Si el síndrome es cero, o no hay error o se ha presentado un error indetectable; la salida NOERROR se establece de acuerdo con ello.

5.9 Comparadores

La comparación de la igualdad de dos palabras binarias es una operación comúnmente utilizada en sistemas de cómputo e interfaces de dispositivos. Por ejemplo, en la figura 2-7(a) de la página 56, mostramos una estructura de sistema en la cual los dispositivo están habilitados al comparar una palabra de "selección de dispositivo" con una "ID de dispositivo" predeterminada. A un circuito que compara dos palabras binarias e indica si son iguales se le conoce como *comparador*. Algunos comparadores interpretan sus palabras de entrada como números con signo o sin signo y también indican una relación aritmética (mayor o igual que) entre las palabras. Estos dispositivos se denominan a menudo *comparadores de magnitud*.

comparador

comparador de magnitud

Tabla 5-49 Programa VHDL de comportamiento para corrección de errores de Hamming.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity hamcorr is
  port (
      DU: IN STD_LOGIC_VECTOR (1 to 7);
      DC: OUT STD_LOGIC_VECTOR (1 to 7);
      NOERROR: OUT STD_LOGIC
);
end hamcorr;

architecture hamcorr of hamcorr is

function syndrome (D: STD_LOGIC_VECTOR)
    return STD_LOGIC_VECTOR is
variable SYN: STD_LOGIC_VECTOR (2 downto 0);
begin
  SYN(0) := D(1) xor D(3) xor D(5) xor D(7);
  SYN(1) := D(2) xor D(3) xor D(6) xor D(7);
  SYN(2) := D(4) xor D(5) xor D(6) xor D(7);
  return(SYN);
end syndrome;

begin
process (DU)
variable i: INTEGER;
  begin
    DC <= DU;
    i := CONV_INTEGER(syndrome(DU));
    if i = 0 then NOERROR <= '1';
    else NOERROR <= '0'; DC(i) <= not DU(i); end if;
  end process;
end hamcorr;
```

5.9.1 Estructura del comparador

Las compuertas OR Exclusivo y NOR Exclusivo pueden visualizarse como comparadores de 1 bit. La figura 5-78(a) ilustra una interpretación de la compuerta XOR 74x86 como un comparador de 1 bit. La salida activa alta, DIFF, es asertiva si las entradas son diferentes. Las salidas de cuatro compuertas XOR son operadas con OR para crear un comparador de 4 bits en (b). La salida DIFF es asertiva si cualquiera de los pares de bits de entrada son diferentes. Con suficientes compuertas XOR y compuertas OR bastante anchas, pueden construirse comparadores con cualquier número de bits de entrada.

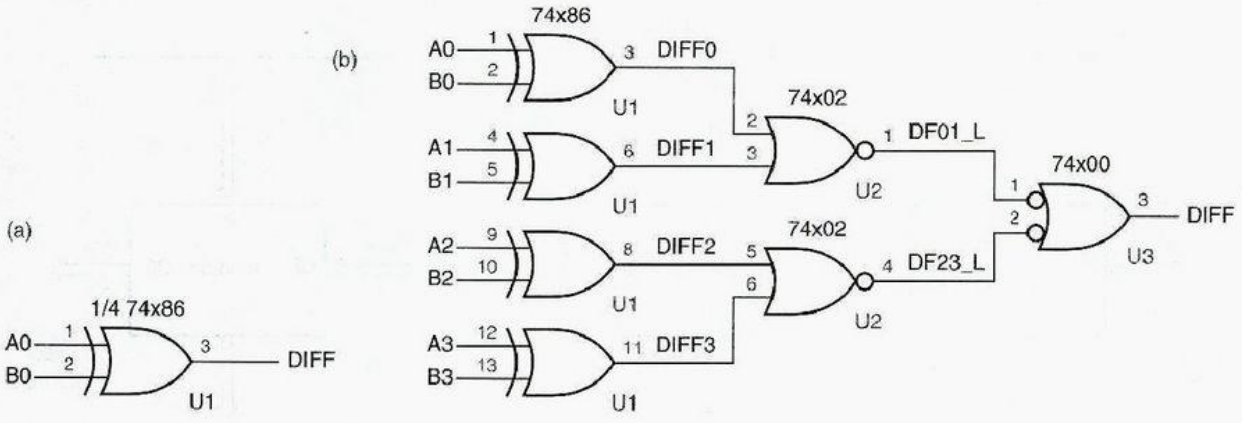


Figura 5-78 Comparadores utilizando el 74x86: a) comparador de 1 bit; b) comparador de 4 bits.

UN COMPARADOR ITERATIVO

Los comparadores de n bits de la subsección anterior pueden denominarse *comparadores paralelos* porque examinan cada par de bits de entrada simultáneamente y liberan los resultados de la comparación de 1 bit en paralelo a una función OR o AND de n entradas. También se puede diseñar un "comparador iterativo" que examine sus bits uno a la vez utilizando una cantidad fija pequeña de lógica por bit. Antes de examinar el diseño del comparador iterativo, debe comprender la clase general de "circuitos iterativos" que se describen en la siguiente subsección.

5.9.2 Circuitos iterativos

Un *circuito iterativo* es un tipo especial de circuito combinacional, cuya estructura es la mostrada en la figura 5-79. El circuito contiene n módulos idénticos, cada uno de los cuales tiene *entradas y salidas primarias*, además de *entradas y salidas en cascada*. Las entradas en cascada más a la izquierda se denominan *entradas de frontera* y se encuentran conectadas a valores lógicos fijos en la mayoría de los circuitos iterativos. Las salidas en cascada más a la derecha se conocen como *salidas de frontera* y por lo regular proporcionan información importante.

circuito iterativo
entradas y salidas primarias
entradas y salidas en cascada
salidas de frontera

Los circuitos iterativos son muy apropiados para problemas que puedan resolverse mediante un algoritmo iterativo simple:

1. Establezca C_0 en su valor inicial y establezca i en 0.
2. Haga uso de C_i y de Pl_i para determinar los valores de PO_i y C_{i+1}.
3. Incremente i.
4. Si $i < n$, vaya al paso 2.

En un circuito iterativo, el ciclo de pasos 2 a 4 se encuentra "desenrollado" al proporcionar un circuito combinacional por separado, que realiza el paso 2 para cada valor de i.

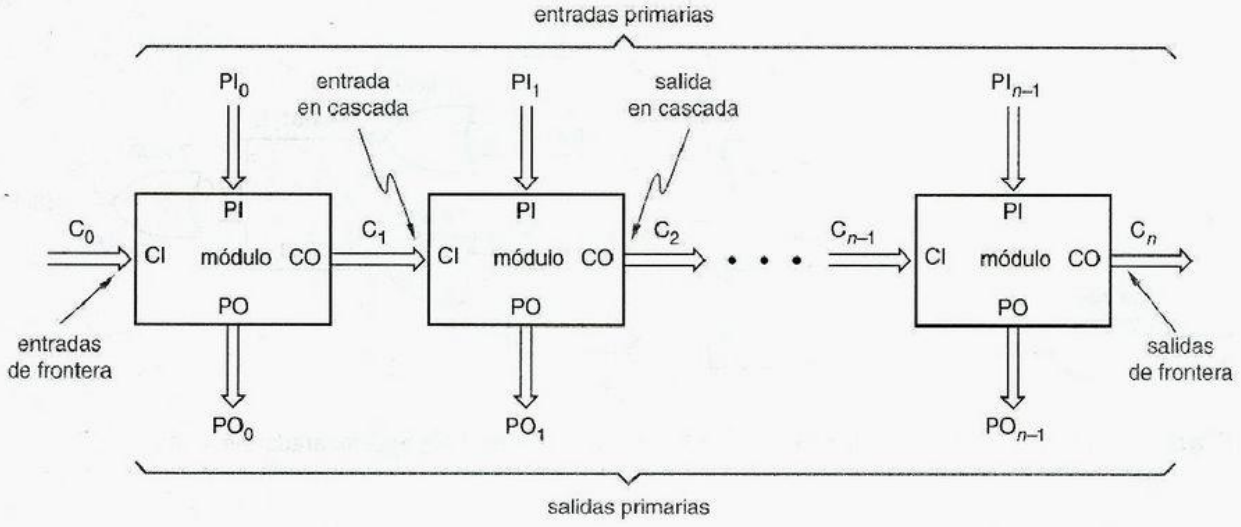


Figura 5-79 Estructura general de un circuito combinacional iterativo.

Ejemplos de circuitos iterativos son el circuito comparador, que se estudia en la subsección siguiente, y el sumador de rizo, de la sección 5.10.2. El comparador de 4 bits 74x85 y el sumador de 4 bits 74x283 son ejemplos de circuitos MSI que se pueden utilizar como los módulos individuales en un circuito iterativo más grande. En la sección 8.6 exploraremos la relación entre los circuitos iterativos y los correspondientes circuitos secuenciales que ejecutan el algoritmo de 4 pasos anterior en pasos discretos de tiempo.

5.9.3 Un circuito comparador iterativo

Dos valores de n bits X y Y se pueden comparar bit por bit utilizando un bit simple EQ_i en cada paso, para seguir la pista de si todos los pares de bits han sido iguales hasta ahora:

1. Establezca EQ_0 en 1 y establezca i en 0.
2. Si EQ_i es 1 y X_i y Y_i son iguales, establezca EQ_{i+1} en 1. Si no, establezca EQ_{i+1} en 0.
3. Incremente i.
4. Si $i < n$, vaya al paso 2.

La figura 5-80 ilustra un circuito iterativo correspondiente. Advierta que este circuito no tiene salidas primarias; la salida de frontera es todo lo que nos interesa. Otro circuito iterativo, como el sumador en rizo de la sección 5.10.2, tiene salidas primarias de interés.

Si puede seleccionar entre el circuito comparador iterativo de esta subsección y uno de los comparadores paralelos que se mostró anteriormente, probablemente preferirá el comparador paralelo. El comparador iterativo implica un ahorro mínimo, si no es que ninguno, y es muy lento a consecuencia de que las señales en cascada necesitan tiempo para "propagarse" desde el módulo más a la izquierda hasta el módulo más a la derecha. Los circuitos iterativos que procesan más de un bit a la vez, utilizando módulos

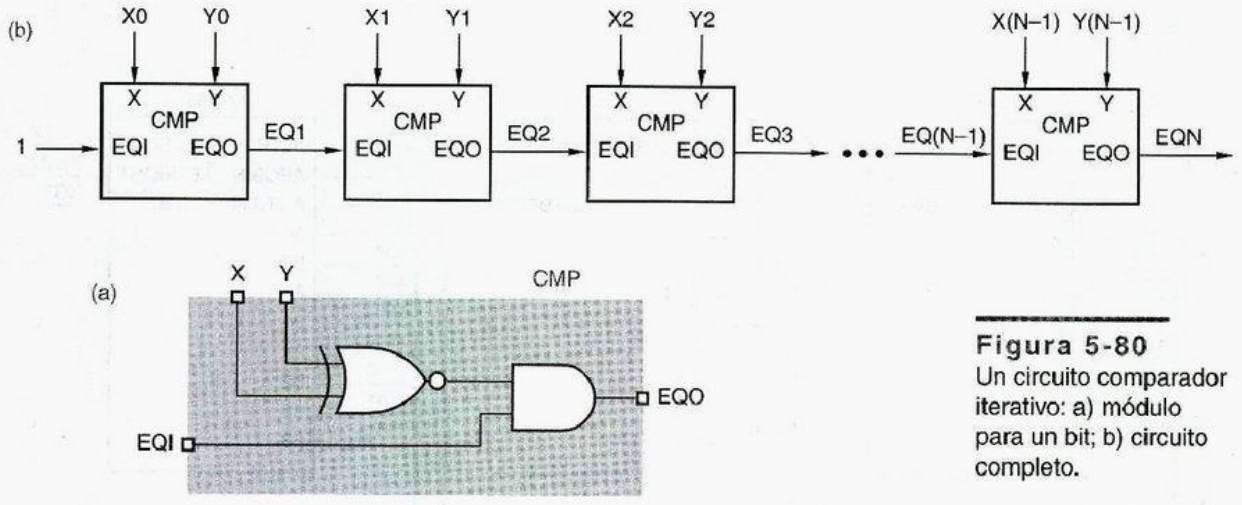


Figura 5-80
Un circuito comparador iterativo: a) módulo para un bit; b) circuito completo.

como el comparador de 4 bits 74x85 y el sumador de 4 bits 74x283, tienen mucho más probabilidad de ser utilizados en diseños prácticos.

5.9.4 Comparadores MSI estándar

Las aplicaciones del comparador son tan comunes que varios comparadores MSI se han desarrollado en forma comercial. El *74x85* es un comparador de 4 bits con el símbolo lógico mostrado en la figura 5-81. Proporciona una salida mayor que (AGTBOUT) y una salida menor que (ALTBOUT), así como también una salida de igualdad (AEQBOUT). El '85 también tiene *entradas en cascada* (AGTBIN, ALTBIN, AEQBIN) para combinar múltiples '85, a fin de crear comparadores para más de 4 bits. Tanto las entradas en cascada como las salidas se encuentran arregladas en un código 1 de 3, puesto que en funcionamiento normal exactamente una entrada y una salida deberían ser asertivas.

74x85

entradas en cascada

Las entradas en cascada se definen de modo que las salidas de un '85 que compara bits menos significativos estén conectadas a las entradas de un '85 que compara los bits

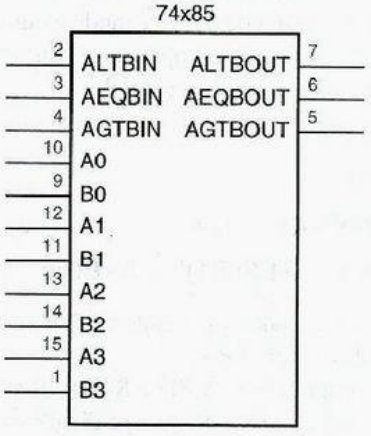


Figura 5-81
Símbolo lógico tradicional para el comparador de 4 bits 74x85.

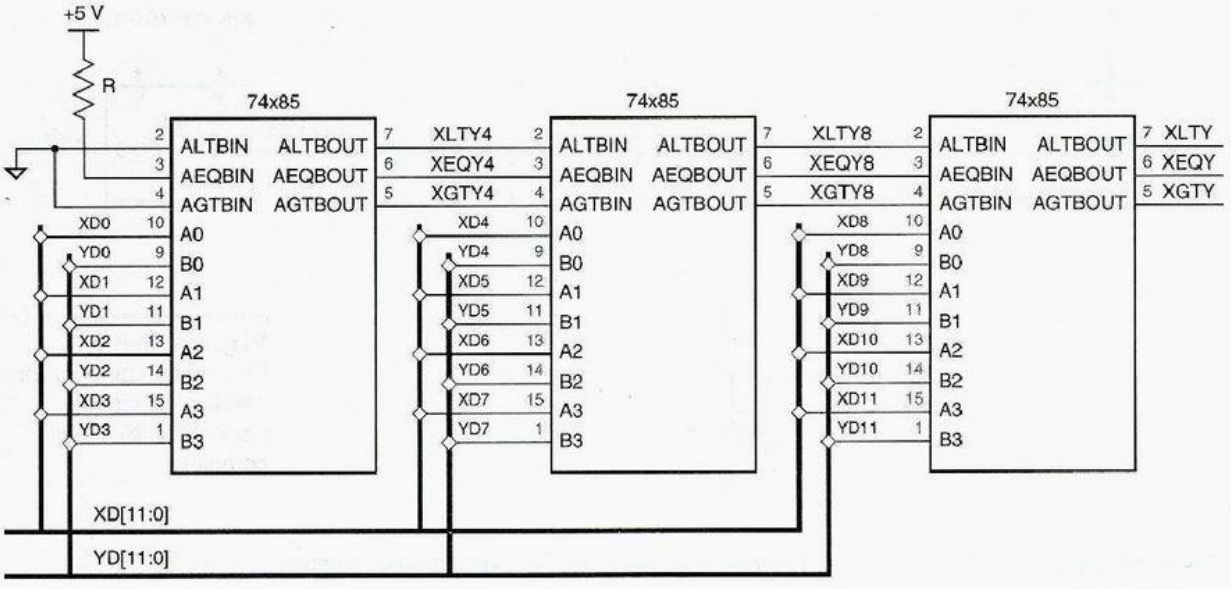


Figura 5-82 Un comparador de 12 bits que utiliza 74x85.

más significativos, como se ilustra en la figura 5-82 para un comparador de 12 bits. Éste es un circuito iterativo de acuerdo con la definición de la sección 5.9.2. Cada '85 desarrolla sus salidas en cascada aproximadamente de acuerdo con las siguientes ecuaciones seudológicas:

$$\text{AGTBOUT} = (A > B) + (A = B) \cdot \text{AGTBIN}$$
$$\text{AEQBOUT} = (A = B) \cdot \text{AEQBIN}$$
$$\text{ALTBOUT} = (A < B) + (A = B) \cdot \text{ALTBIN}$$

Las anteriores subexpresiones entre paréntesis no son expresiones lógicas normales: en vez de ello, indican una comparación aritmética que se presenta entre las entradas A3-A0 y B3-B0. En otras palabras, AGTBOUT es asertiva si A > B o si A = B y AGTBIN es asertiva (si los bits de orden mayor son iguales, tenemos que observar los bits de orden menor para la respuesta). Veremos esta clase de expresión de nuevo cuando examinemos el diseño del comparador ABEL en la sección 5.9.5. Las comparaciones aritméticas pueden expresarse utilizando expresiones lógicas normales, por ejemplo,

$$\begin{aligned}(A > B) = {} & A3 \cdot B3' + \\ & (A3 \oplus B3)' \cdot A2 \cdot B2' + \\ & (A3 \oplus B3)' \cdot (A2 \oplus B2)' \cdot A1 \cdot B1' + \\ & (A3 \oplus B3)' \cdot (A2 \oplus B2)' \cdot (A1 \oplus B1)' \cdot A0 \cdot B0'\end{aligned}$$

Tales expresiones deben sustituirse en las ecuaciones seudológicas anteriores para obtener ecuaciones lógicas genuinas para las salidas del comparador.

También se tienen disponibles diversos comparadores MSI de 8 bits. El más simple es el *74x682*, cuyo símbolo lógico se ilustra en la figura 5-83 y cuyo diagrama lógico

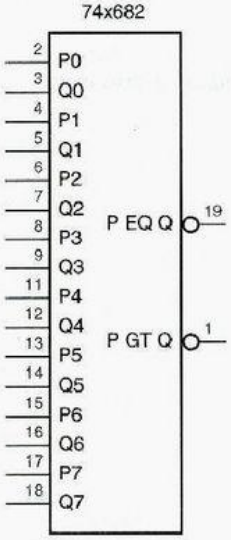


Figura 5-83 Símbolo lógico tradicional para el comparador de 8 bits 74x682.

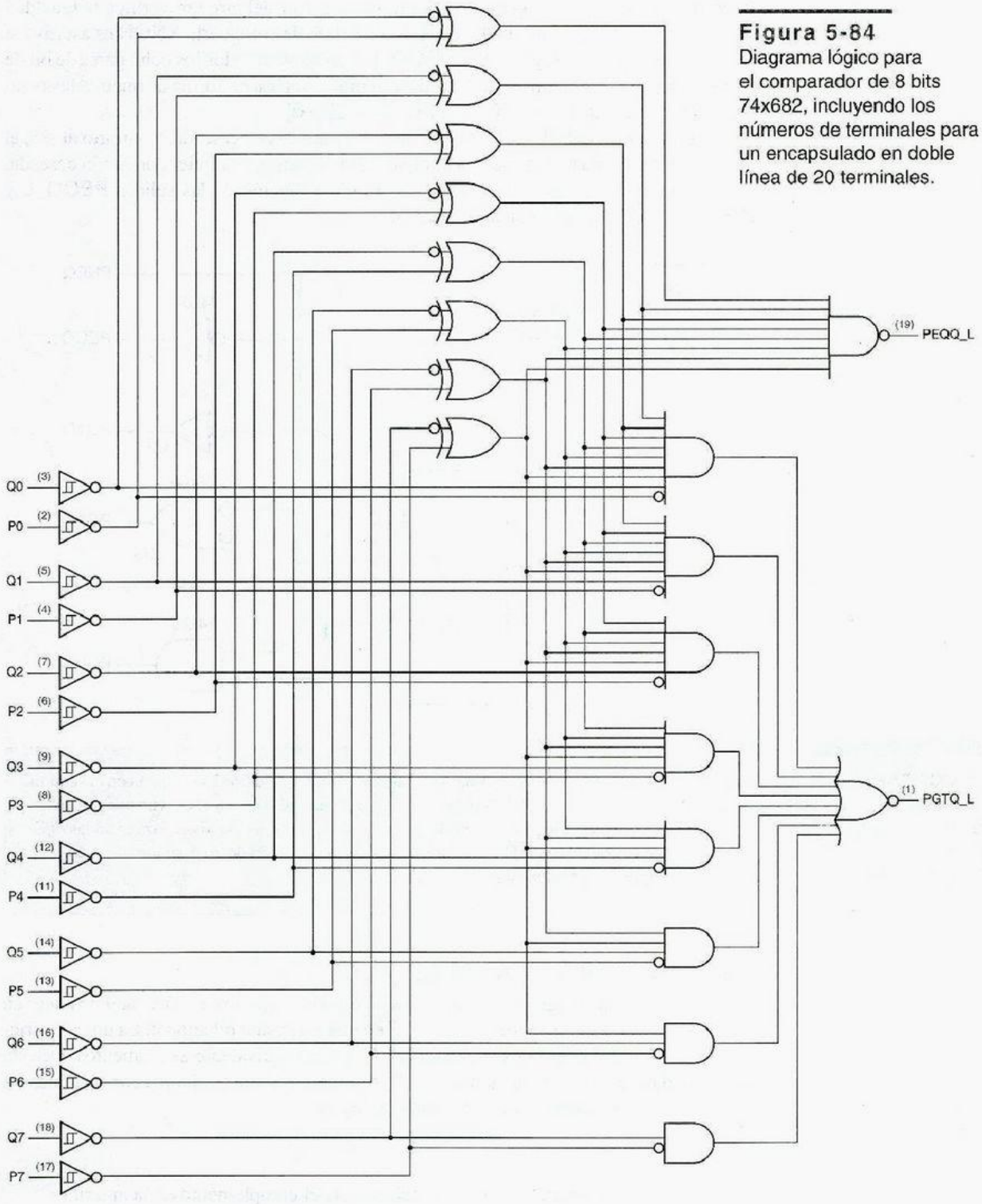
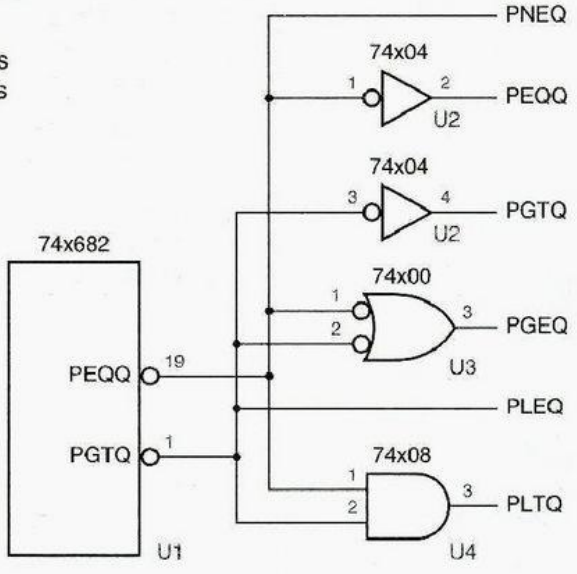


Figura 5-84
Diagrama lógico para el comparador de 8 bits 74x682, incluyendo los números de terminales para un encapsulado en doble línea de 20 terminales.

interno se presenta en la figura 5-84. La mitad superior del circuito verifica la igualdad de las dos palabras de entrada de 8 bits. Cada salida de compuerta XNOR es asertiva si sus entradas son iguales, y la salida PEQQ_L es asertiva si todos los ocho pares de bit de entrada son iguales. La mitad inferior del circuito compara en forma aritmética las palabras de entrada y afirma PGTQ_L si P[7–0] > Q[7–0].

A diferencia del 74x85, el 74x682 no tiene entradas en cascada. Contrario al '85, el '682 no proporciona una salida "menor que". Sin embargo, cualquier condición deseada, incluyendo ≤ y ≥, puede ser formulada como una función de las salidas PEQQ_L y PGTQ_L, como se ilustra en la figura 5-85.

Figura 5-85
Condiciones aritméticas derivadas de las salidas del 74x682.



COMPARACIÓN DE COMPARADORES

Los comparadores individuales de 1 bit (compuertas XNOR) en el '682 se dibujan en el sentido opuesto al de los ejemplos de la subsección anterior: las salidas son asertivas para entradas *iguales* y posteriormente se operan con AND, en lugar de que sean asertivas para entradas *diferentes* y luego operadas con OR. Podemos examinar una función de comparador de cualquier manera, siempre que seamos consistentes.

5.9.5 Comparadores en ABEL y PLD

La comparación de la igualdad o desigualdad en dos conjuntos es muy fácil de hacer en ABEL si se utiliza el operador "==" o "!=" en una expresión relacional. La única restricción es que los dos conjuntos deben tener un número equivalente de elementos. De esta forma, dada la expresión relacional "A!=B", donde A y B son conjuntos con *n* elementos cada uno, el compilador genera la expresión lógica

```
(A1 $ B1)  # (A2 $ B2)  # ... # (An $ Bn)
```

La expresión lógica para "A==B" es precisamente el complemento de la anterior.

En la expresión lógica anterior, la comparación de cada bit toma una función XOR de 2 entradas. Puesto que una función XOR de 2 entradas se puede realizar como una suma de dos términos de producto, la expresión completa puede ser efectuada en un PLD como una suma relativamente modesta de $2n$ términos de producto:

```
(A1&!B1 # !A1&B1) # (A2&!B2 # !A2&&B2) # ... # (An&!Bn # !An&&Bn)
```

Aunque ABEL tiene operadores relacionales para comparaciones menor que y mayor que, las expresiones lógicas resultantes no son tan pequeñas o fáciles de derivar. Por ejemplo, considere la expresión relacional "A<B", donde `[An..A1]` y `[Bn..B1]` son conjuntos con n elementos. Para construir la expresión lógica correspondiente, ABEL primero construye n ecuaciones de la forma

```
Li = (!Ai & (Bi # Li-1) # (Ai & Bi & Li-1)
```

para $i = 1$ hasta n y `L0` = 0 por definición. Ésta es, en efecto, una definición iterativa de la función menor que, comenzando con el bit menos significativo. Cada ecuación `Li` dice que, desde el bit i, `A` es menor que `B` si `Ai` es 0 y `Bi` es 1 o `A` era menor que `B` desde el bit anterior, o si `Ai` y `Bi` son ambos 1 y `A` era menor que `B` desde el bit anterior.

La expresión lógica para "A<B" es sencillamente la ecuación para `Ln`. De este modo, después de crear las n ecuaciones anteriores, ABEL las colapsa en una ecuación simple para `Ln` involucrando solamente elementos de `A` y de `B`. Se hace esto mediante la sustitución de la ecuación `Ln-1` en el lado derecho de la ecuación `Ln`, sustituyendo luego la ecuación `Ln-2` en este resultado, y así sucesivamente, hasta sustituir 0 por `L0`. Finalmente, se deriva una expresión mínima de suma de productos del resultado.

El colapso de un circuito iterativo en una realización de producto de sumas de dos niveles crea una expansión exponencial de términos de producto. La función de comparación " < " sigue este patrón, requiriendo $2^n - 1$ términos de producto para un comparador de n bits. De esta forma, los comparadores mayores de unos cuantos bits no se pueden realizar en forma práctica (en un solo paso) usando un PLD.

Los resultados para comparadores ">" son idénticos, naturalmente, y las expresiones lógicas para ">=" y "<=" son, por lo menos, tan malas, siendo los complementos de las expresiones para "<" y ">". Si empleamos un PLD con control de polaridad de salida, la inversión es libre y el número de términos de producto es el mismo; de otro modo, el número mínimo de términos de producto después de la inversión es de $2^n + 2^{n-1} - 1$.

5.9.6 Comparadores en VHDL

VHDL tiene operadores de comparación para todos sus tipos integrados. Los operadores de *igualdad (=)* y *desigualdad (/=)* se aplican a todos los tipos; para tipos arreglo y registro, los operandos deben tener igual tamaño y estructura, y los operandos se comparan componente por componente. En muchos ejemplos de este capítulo, hemos utilizado el operador de igualdad para comparar una señal o vector de señal con un valor constante. Si comparamos dos señales o variables, el motor de síntesis genera ecuaciones semejantes a las de ABEL en la subsección anterior.

igualdad, =
desigualdad, /=

Los otros operadores de comparación de VHDL, >, <, >=, y <= se aplican solamente a tipos enteros, tipos enumerados (tales como STD_LOGIC) y arreglos unidimensionales de tipos enteros o de enumeración. El orden entero del más pequeño al más grande es el orden natural, de menos infinito a más infinito, y los tipos enumerados emplean el orden en el que los elementos del tipo fueron definidos, desde el primero hasta el último (a menos que explícitamente se modifique la codificación de enumeración empleando un comando específico para el motor de síntesis, en cuyo caso la ordenación es la de esa codificación).

La ordenación para los tipos arreglo es definida de manera iterativa, comenzando con el elemento *más a la izquierda* en cada arreglo. Los arreglos siempre se comparan de izquierda a derecha, sin tener en cuenta el orden de sus intervalos de índice ("to" o "downto"). El orden del par más a la izquierda de elementos desiguales es el orden del arreglo. Si los arreglos tienen longitudes distintas, y todos los elementos del arreglo más corto coinciden con los elementos correspondientes del más extenso, entonces el arreglo más corto se considera el menor.

El resultado de todo esto es que los operadores de comparación integrados comparan arreglos de igual longitud del tipo BIT_VECTOR o STD_LOGIC_VECTOR como si representaran enteros sin signo. Si los arreglos tienen diferentes longitudes, entonces los operadores *no* producen una comparación aritmética válida, la que se obtendría mediante la extensión del arreglo más corto con ceros hacia la izquierda; hablaremos más de esto en un momento.

La tabla 5-50 es un programa VHDL que produce todas las salidas de comparación para comparar dos enteros sin signo de 8 bits. Puesto que los dos vectores de entrada A y B tienen longitudes iguales, el programa produce los resultados deseados.

Tabla 5-50
Programa VHDL de comportamiento para comparación de enteros sin signo de 8 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity vcompare is
    port (
        A, B: in STD_LOGIC_VECTOR (7 downto 0);
        EQ, NE, GT, GE, LT, LE: out STD_LOGIC
    );
end vcompare;

architecture vcompare_arch of vcompare is
begin
process (A, B)
  begin
    EQ <= '0'; NE <= '0'; GT <= '0'; GE <= '0'; LT <= '0'; LE <= '0';
    if A = B then EQ <= '1'; end if;
    if A /= B then NE <= '1'; end if;
    if A > B then GT <= '1'; end if;
    if A >= B then GE <= '1'; end if;
    if A < B then LT <= '1'; end if;
    if A <= B then LE <= '1'; end if;
  end process;
end vcompare_arch
```

Para permitir operaciones aritméticas y comparaciones más flexibles, el estándar IEEE 1076-3 creó un paquete estándar, `std_logic_arith`, el cual define dos nuevos tipos importantes y un montón de funciones aritméticas y de comparación que operan en ellos. Los dos nuevos tipos son `SIGNED` y `UNSIGNED`:

```
type SIGNED is array (NATURAL range <>) of STD_LOGIC;
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
```

Como se puede ver, ambos tipos se encuentran definidos precisamente como arreglos de longitud indeterminada de `STD_LOGIC`, sin ser diferentes de `STD_LOGIC_VECTOR`. El punto importante es que el paquete también define nuevas funciones de comparación que son invocadas cuando alguno o ambos operandos de comparación tienen uno de los nuevos tipos. Por ejemplo, define ocho nuevas funciones "menor-que" con las siguientes combinaciones de parámetros:

```
function "<" (L: UNSIGNED; R: UNSIGNED) return BOOLEAN;
function "<" (L: SIGNED; R: SIGNED) return BOOLEAN;
function "<" (L: UNSIGNED; R: SIGNED) return BOOLEAN;
function "<" (L: SIGNED; R: UNSIGNED) return BOOLEAN;
function "<" (L: UNSIGNED; R: INTEGER) return BOOLEAN;
function "<" (L: INTEGER; R: UNSIGNED) return BOOLEAN;
function "<" (L: SIGNED; R: INTEGER) return BOOLEAN;
function "<" (L: INTEGER; R: SIGNED) return BOOLEAN;
```

De este modo, el operador "<" puede utilizarse con cualquier combinación de los operandos `SIGNED`, `UNSIGNED` e `INTEGER`; el compilador selecciona la función cuyos tipos de parámetro satisfagan los operandos reales. Cada una de las funciones se define en el paquete para que haga "lo correcto", incluyendo la realización de conversiones y extensiones apropiadas, cuando se utilizan operandos de diferentes tamaños o tipos. Los otros cinco operadores relacionales, `=`, `/=`, `<=`, `>`, y `>=`, proporcionan funciones semejantes.

Con el paquete `IEEE_std_logic_arith`, se pueden escribir programas como el de la tabla 5-51. Sus ocho vectores de entrada de 8 bits, `A`, `B`, `C`, y `D`, tienen tres tipos diferentes. En las comparaciones que involucran `A`, `B`, y `C`, el compilador automáticamente selecciona la versión correcta de la función de comparación; por ejemplo, para "`A<B`" selecciona la primera función "<" anterior, porque ambos operandos tienen tipo `UNSIGNED`.

En las comparaciones que involucran a `D`, se emplean conversiones de tipo explícitas. La suposición es que el diseñador quiere que este particular `STD_LOGIC_VECTOR` sea interpretado como `UNSIGNED`, en un caso, y como `SIGNED`, en el otro. El punto de importancia que hay que comprender aquí es que el paquete `std_logic_arith` no hace suposición alguna acerca de cómo se interpretarán los `STD_LOGIC_VECTOR`s; el usuario debe especificar la conversión.

Otros dos paquetes, `std_logic_signed` y `std_logic_unsigned`, hacen suposiciones y son útiles si todos los `STD_LOGIC_VECTOR`s serán interpretados de la misma manera. Cada paquete contiene tres versiones de cada función de comparación de modo que los `STD_LOGIC_VECTOR`s se interpretan como `SIGNED` o `UNSIGNED`, respectivamente, cuando se comparan con cada uno de los otros o con enteros.

Tabla 5-51 Programa VHDL de comportamiento para comparación de enteros de 8 bits de diversos tipos.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity vcompa is
    port (
        A, B: in UNSIGNED (7 downto 0);
        C: in SIGNED (7 downto 0);
        D: in STD_LOGIC_VECTOR (7 downto 0);
        A_LT_B, B_GE_C, A_EQ_C, C_NEG, D_BIG, D_NEG: out STD_LOGIC
    );
end vcompa;

architecture vcompa_arch of vcompa is
begin
process (A, B, C, D)
  begin
    A_LT_B <= '0'; B_GE_C <= '0'; A_EQ_C <= '0'; C_NEG <= '0'; D_BIG <= '0'; D_NEG <= '0';
    if A < B then A_LT_B <= '1'; end if;
    if B >= C then B_GE_C <= '1'; end if;
    if A = C then A_EQ_C <= '1'; end if;
    if C < 0 then C_NEG <= '1'; end if;
    if UNSIGNED(D) > 200 then D_BIG <= '1'; end if;
    if SIGNED(D) < 0 then D_NEG <= '1'; end if;
  end process;
end vcompa_arch;
```

Cuando una función de comparación se especifica en VHDL, la realización de la misma como un producto de sumas de dos niveles toma los mismos términos de producto que en ABEL. Sin embargo, la mayoría de las máquinas de síntesis de VHDL realizarán el comparador como un circuito iterativo con bastante menos compuertas, aunque más niveles de lógica. Asimismo, mejores máquinas de síntesis pueden detectar oportunidades para eliminar circuitos comparadores completos. Por ejemplo, en el programa de la tabla 5-50 de la página 428, las salidas NE, GE, y LE podrían realizarse con un inversor cada una, como los complementos de las salidas EQ, LT, y GT, respectivamente.

*5.10 Sumadores, restadores y ALU

sumador

La suma es la operación aritmética que se realiza con mayor frecuencia en los sistemas digitales. Un *sumador* combina dos operandos aritméticos que utilizan las reglas de suma que se describen en el capítulo 2. Como demostramos en la sección 2.6, las mismas reglas de suma y por consiguiente los mismos sumadores, se emplean tanto para números sin signo como para los de complemento a dos. Un sumador puede efectuar la resta como la suma del minuendo y el sustraendo complementado (negado), pero también es posible construir circuitos *restadores* que hagan la resta directamente. Los dispositivos MSI denominados ALU, descritos en la sección 5.10.6, efectúan la suma, la

restador

resta o cualquier operación de acuerdo con un código de operación que se suministra al dispositivo.

*5.10.1 Medio sumadores y sumadores completos

medio sumador

El sumador más simple, conocido como un medio sumador, suma dos operandos de 1 bit X y Y, y produce una suma de 2 bits. La suma puede abarcar desde el 0 hasta el 2, lo cual requiere de dos bits para expresarse. El bit de menor orden de la suma puede nombrarse HS (suma media, por sus siglas en inglés) y el bit de orden mayor puede llamarse CO (acarreo de salida, por sus siglas en inglés). Podemos escribir las ecuaciones siguientes para HS y CO:

$$\begin{aligned} HS &= X + Y \\ &= X \cdot Y' + X' \cdot Y \\ CO &= X \cdot Y \end{aligned}$$

Para sumar operandos con más de un bit, debemos prever acarreos entre posiciones de bits. El bloque de construcción para esta operación se conoce como un *sumador completo*. Aparte de las entradas X y Y de bit de sumandos, un sumador completo tiene un bit de acarreo de entrada, CIN. La suma de las tres entradas puede abarcar de 0 a 3, lo que aún puede expresarse con sólo dos bits de salida, S y COUT, teniendo las ecuaciones siguientes:

$$\begin{aligned} S &= X \oplus Y \oplus CIN \\ &= X \cdot Y' \cdot CIN' + X' \cdot Y \cdot CIN' + X' \cdot Y' \cdot CIN + X \cdot Y \cdot CIN \\ COUT &= X \cdot Y + X \cdot CIN + Y \cdot CIN \end{aligned}$$

Aquí, S es 1 si un número impar de las entradas es 1, y COUT es 1 si dos o más de las entradas son 1. Estas ecuaciones representan la misma operación que fue especificada mediante la tabla de suma binaria en la tabla 2-3 de la página 32.

Un circuito posible que realiza las ecuaciones del sumador completo se muestra en la figura 5-86(a). El símbolo lógico correspondiente se ilustra en (b). En ocasiones, el símbolo se dibuja como se muestra en (c), de modo que se pueden dibujar sumadores completos en cascada más cuidadosamente, como en la subsección que sigue.

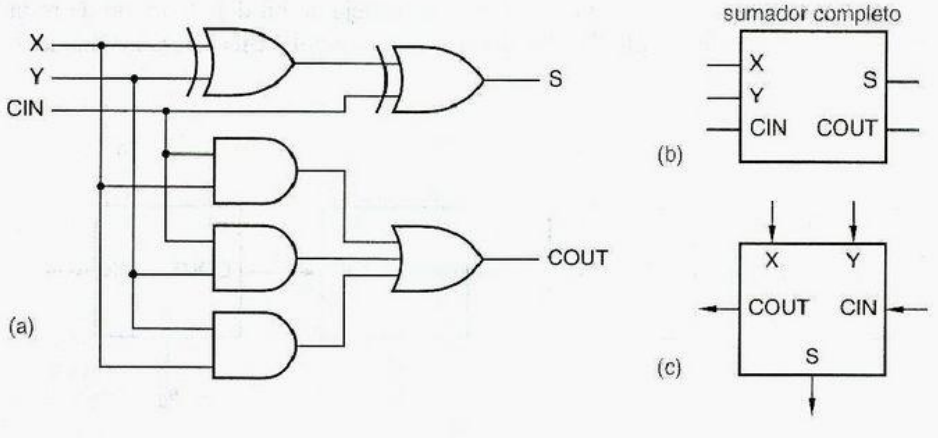


Figura 5-86
Sumador completo: (a) diagrama de circuito al nivel de compuerta; (b) símbolo lógico; (c) símbolo lógico alterno adecuado para representación en cascada.

*5.10.2 Sumadores iterativos ("en rizo")

sumador iterativo ("en rizo")

Dos palabras binarias, con n bits cada una, pueden sumarse con un *sumador iterativo*: una cascada de n etapas de sumadores completos, cada una de las cuales maneja un bit. La figura 5-87 ilustra el circuito para un sumador iterativo de 4 bits. La entrada de acarreo para el bit menos significativo (c_0) se establece normalmente a 0 y la salida de acarreo de cada sumador completo se conecta a la entrada de acarreo del siguiente sumador completo más significativo. El sumador iterativo es un ejemplo clásico de un circuito iterativo, como se definió en la sección 5.9.2.

Un sumador iterativo es lento, puesto que en el peor de los casos un acarreo debe propagarse desde el sumador completo menos significativo hasta el más significativo. Esto ocurre si, por ejemplo, un sumando es 11 ... 11 y el otro es 00 ... 01. Suponiendo que todos los bits del sumando se presenten de manera simultánea, el retardo, en el peor de los casos, es

$$t_{ADD} = t_{XYCout} + (n - 2) \cdot t_{CinCout} + t_{CinS}$$

donde t_{XYCout} es el retardo desde X hasta Y para COUT, en la etapa menos significativa; $t_{CinCout}$ es el retardo desde CIN hasta COUT, en las etapas intermedias, y t_{CinS} es el retardo desde CIN hasta S, en la etapa más significativa.

Un sumador rápido puede construirse al obtener cada salida de suma s_i con sólo dos niveles de lógica. Esto puede llevarse a cabo al escribir una ecuación para s_i en términos de $x_0 - x_i$, $y_0 - y_i$ y c_0, "expandiendo" la multiplicación o la suma para obtener una expresión de producto de sumas, y construyendo el circuito correspondiente AND-OR u OR-AND. Desgraciadamente, más allá de s_2 las expresiones resultantes tienen demasiados términos, requiriendo de demasiadas compuertas de primer nivel y más entradas de las que por lo regular son posibles en la compuerta de segundo nivel. Por ejemplo, aún suponiendo que $c_0 = 0$, un circuito AND–OR de dos niveles para s_2 requiere 14 AND de 4 entradas, cuatro AND de 5 entradas y una compuerta OR de 18 entradas; los bits de suma de mayor orden son todavía peores. No obstante, es posible construir sumadores con solamente unos cuantos niveles de retardo utilizando un número más razonable de compuertas, como veremos en la sección 5.10.4.

*5.10.3 Sustractores (restadores)

restador completo

Una operación de resta binaria análoga a la suma binaria también se especificó en la tabla 2-3 de la página 32. Un *restador completo* maneja un bit del algoritmo de resta binaria, teniendo bits de entrada X (minuendo), Y (sustraendo) y BIN (préstamo entrada),

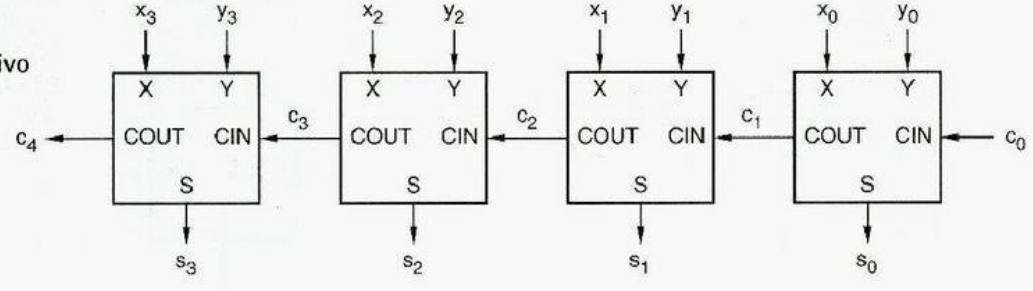


Figura 5-87
Un sumador iterativo de 4 bits.

y bits de salida D (diferencia) y BOUT (préstamo salida). Podemos escribir las ecuaciones lógicas correspondientes a la tabla de resta binaria como sigue:

$$D = X \oplus Y \oplus BIN$$

$$BOUT = X' \cdot Y + X' \cdot BIN + Y \cdot BIN$$

Estas ecuaciones son muy semejantes a las ecuaciones para un sumador completo, lo que no debería sorprendernos. Demostramos en la sección 2.6 que una operación de resta de complemento a dos, $X - Y$, puede realizarse mediante una operación de suma, a saber, sumando el complemento a dos de Y a X. El complemento a dos de Y es $\overline{Y} + 1$, donde $\overline{Y}$ es el complemento de bit a bit de Y. También demostramos en el ejercicio 2.26 que un sumador binario puede emplearse para efectuar una operación de resta sin signo $X - Y$, al realizar la operación $X + \overline{Y} + 1$. Confirmamos que estas instrucciones son verdaderas al manipular las ecuaciones lógicas anteriores:

$$BOUT = X' \cdot Y + X' \cdot BIN + Y \cdot BIN$$

$$BOUT' = (X + Y') \cdot (X + BIN') \cdot (Y' + BIN') \quad \text{teorema generalizado de DeMorgan}$$

$$= X \cdot Y' + X \cdot BIN' + Y' \cdot BIN' \quad \text{distribución de la multiplicación}$$

$$D = X \oplus Y \oplus BIN$$

$$= X \oplus Y' \oplus BIN' \quad \text{complementación de entradas XOR}$$

Para la última manipulación, recordemos que podemos complementar las dos entradas de una compuerta XOR sin cambiar la función realizada.

En comparación con las ecuaciones para un sumador completo, las ecuaciones anteriores nos dicen que podemos construir un restador completo a partir de un sumador completo, como se ilustra en la figura 5-88. Sólo por razones de claridad, hemos dado un nombre ficticio al circuito sumador completo en (a) el "74x999". Como se observa en (c), podemos interpretar la función de este mismo circuito físico como la de un restador completo dándole un nuevo símbolo con señales de sustraendo, préstamo entrada, préstamo salida activas bajas.

De esta forma, para construir un restador iterativo para dos operandos activos altos de n bits, podemos utilizar n 74x999 e inversores, como se muestra en (d). Advierta que para la operación de resta, la entrada de préstamo del bit menos significativo debería ser negada (no préstamos), lo que significa para una entrada activa baja que la terminal física debe ser 1 o ALTO. Éste es justo el caso opuesto al de la suma, donde la misma terminal de entrada es un acarreo de entrada activo alto que es 0 o BAJO.

Al retroceder a las matemáticas del capítulo 2, podemos demostrar que esta clase de manipulación funciona para todo circuito sumador y restador, no solamente para sumadores y restadores iterativos. Es decir, cualquier circuito sumador de n bits puede hacerse funcionar como un restador al complementar el sustraendo y al tratar las señales de acarreo de entrada y acarreo de salida como préstamos con el nivel activo opuesto. El resto de esta sección trata sólo sobre circuitos de suma, en el entendido de que se modifican con facilidad para efectuar la resta.

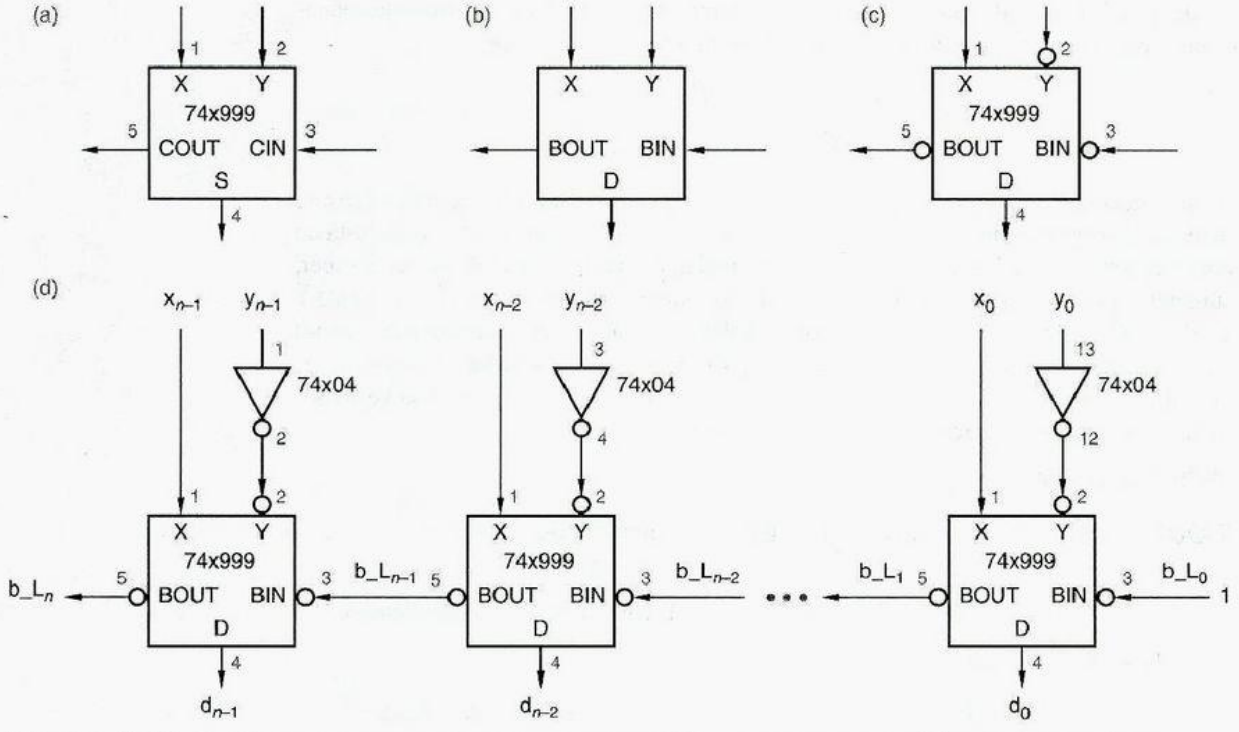


Figura 5-88 Diseño de restadores utilizando sumadores: (a)sumador completo; (b) restador completo; (c) interpretación del dispositivo en: (a) como un sustractor completo; (d) sustractor en rizo.

*5.10.4 Sumadores de acarreo anticipado

La ecuación lógica para sumar el bit i de un sumador binario puede en realidad escribirse de manera bastante simple:

$$s_i = x_i \oplus y_i \oplus c_i$$

Se introduce más complejidad cuando expandimos la c_i anterior en términos de $x_0 - x_{i-1}$, $y_0 - y_{i-1}$, y c_0, y obtenemos un verdadero desorden al expandir los XOR. Sin embargo, si deseamos renunciar a la expansión, podemos al menos racionalizar el diseño de la lógica de c_i usando ideas de *acarreo anticipado*, discutidas en esta subsección.

acarreo anticipado

La figura 5-89 muestra la idea básica. El bloque denominado "Lógica de Acarreo anticipado" calcula c_i en un número fijo y pequeño de niveles lógicos para cualquier valor razonable de i. Dos definiciones son la clave para la lógica de acarreo anticipado:

- Para una combinación particular de entradas x_i y y_i, se dice que la etapa i de sumador *genera* un acarreo si produce un acarreo de salida de 1 ($c_{i+1} = 1$) independiente de las entradas en $x_0 - x_{i-1}$, $y_0 - y_{i-1}$, y c_0.

generación de acarreo

- Para una combinación particular de entradas x_i y y_i, se dice que la etapa i de sumador *propaga* el acarreo si produce un acarreo de salida de 1 ($c_{i+1} = 1$) en la presencia de una combinación de entrada de $x_0 - x_{i-1}$, $y_0 - y_{i-1}$, y c_0 que provoca un acarreo de entrada de 1 ($c_i = 1$).

propagación de acarreo

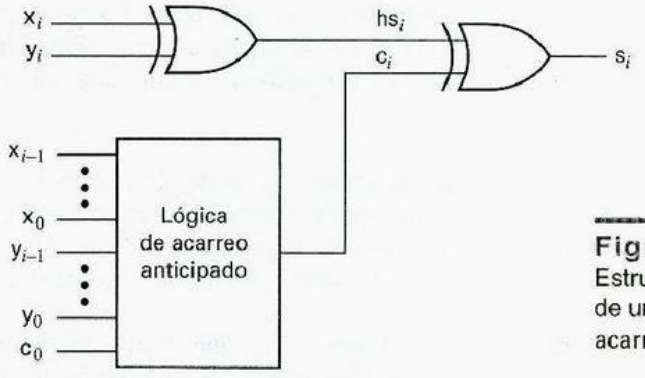


Figura 5-89
Estructura de una etapa de un sumador de acarreo anticipado.

En correspondencia con estas definiciones, escribimos las ecuaciones lógicas para una señal generada de acarreo, g_i, y una señal propagada de acarreo, p_i, para cada etapa de un sumador de acarreo anticipado:

$$g_i = x_i \cdot y_i$$
$$p_i = x_i + y_i$$

Esto es, una etapa genera incondicionalmente un acarreo si tanto uno como el otro de sus bits de sumando son 1, y propaga acarreos si por lo menos uno de sus bits de sumando es 1. La salida de acarreo de una etapa puede ahora escribirse en términos de las señales generada y propagada:

$$c_{i+1} = g_i + p_i \cdot c_i$$

Para eliminar el rizo de acarreo, expandimos de manera recursiva el término c_i para cada etapa y distribuimos la multiplicación para obtener una expresión AND-OR de dos niveles. Empleando esta técnica, podemos obtener las siguientes ecuaciones de acarreo para las primeras cuatro etapas de sumador:

$$\begin{aligned}
c_1 &= g_0 + p_0 \cdot c_0 \\
c_2 &= g_1 + p_1 \cdot c_1 \\
&= g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0) \\
&= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0 \\
c_3 &= g_2 + p_2 \cdot c_2 \\
&= g_2 + p_2 \cdot (g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0) \\
&= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0 \\
c_4 &= g_3 + p_3 \cdot c_3 \\
&= g_3 + p_3 \cdot (g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0) \\
&= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0
\end{aligned}$$

sumador de acarreo anticipado

Cada ecuación corresponde a un circuito con apenas tres niveles de retardo; uno para las señales generada y propagada, y dos para el producto de sumas mostrado. Un *sumador de acarreo anticipado* utiliza ecuaciones de tres niveles como éstas, en cada etapa de sumador, para el bloque etiquetado "acarreo anticipado", en la figura 5-89. La salida de

suma para una etapa es producida al combinar el bit de acarreo con los dos bits de sumando para la etapa, como lo mostramos en la figura. En la subsección siguiente estudiaremos algunos sumadores MSI comerciales y los ALU que utilizan el acarreo anticipado.

*5.10.5 Sumadores MSI

74x283

74x83

El *74x283* es un sumador binario de 4 bits que forma sus salidas de acarreo, y suma con sólo unos cuantos niveles de lógica, utilizando la técnica del acarreo anticipado. La figura 5-90 es un símbolo lógico para el 74x283. El antiguo *74x83* es casi idéntico, excepto por sus terminales de salida, las cuales tienen ubicaciones no estándar para la alimentación y la tierra.

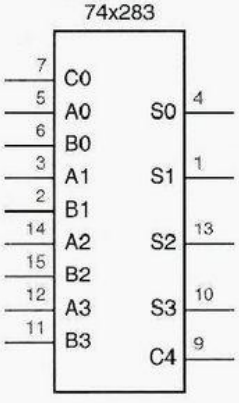


Figura 5-90
Símbolo lógico tradicional para el sumador binario de 4 bits 74x283.

El diagrama lógico para el '283, que se muestra en la figura 5-91, tiene apenas algunas diferencias con respecto al diseño general de acarreo anticipado que describimos en la subsección anterior. En primer lugar, sus sumandos se denominan A y B en lugar de X y Y; nada de otro mundo. En segundo, produce versiones activas bajas de las señales generada de acarreo (g_i') y propagada de acarreo (p_i'), puesto que las compuertas invertidas son generalmente más rápidas que las no invertidas. En tercero, se aprovecha del hecho de que podemos manipular en forma algebraica la ecuación de suma media como sigue:

$$\begin{aligned} hs_i &= x_i \oplus y_i \\ &= x_i \cdot y_i' + x_i' \cdot y_i \\ &= x_i \cdot y_i' + x_i \cdot x_i' + x_i' \cdot y_i + y_i \cdot y_i' \\ &= (x_i + y_i) \cdot (x_i' + y_i') \\ &= (x_i + y_i) \cdot (x_i \cdot y_i)' \\ &= p_i \cdot g_i' \end{aligned}$$

De este modo, una compuerta AND con una entrada invertida puede utilizarse en lugar de una compuerta XOR para crear cada bit de la suma media.

Finalmente, el '283 crea las señales de acarreo empleando una estructura INVERSOR-OR-AND (el equivalente de DeMorgan de un AND-OR-INVERSOR), el cual tiene aproximadamente el mismo retardo que una compuerta de inversión CMOS o TTL simples. Esto requiere una breve explicación, puesto que las ecuaciones de acarreo que derivamos en la subsección anterior son empleadas en una forma ligeramente modificada. En particular, la ecuación c_{i+1} utiliza el término $p_i \cdot g_i$ en vez de g_i. Esto no tiene efecto sobre la salida, puesto que p_i es siempre 1 cuando g_i es 1. Sin embargo, permite que la ecuación sea factorizada como sigue:

$$\begin{aligned} c_{i+1} &= p_i \cdot g_i + p_i \cdot c_i \\ &= p_i \cdot (g_i + c_i) \end{aligned}$$

Esto conduce a las siguientes ecuaciones de acarreo, que son utilizadas por el circuito:

$$\begin{aligned} c_1 &= p_0 \cdot (g_0 + c_0) \\ c_2 &= p_1 \cdot (g_1 + c_1) \\ &= p_1 \cdot (g_1 + p_0 \cdot (g_0 + c_0)) \\ &= p_1 \cdot (g_1 + p_0) \cdot (g_1 + g_0 + c_0) \end{aligned}$$

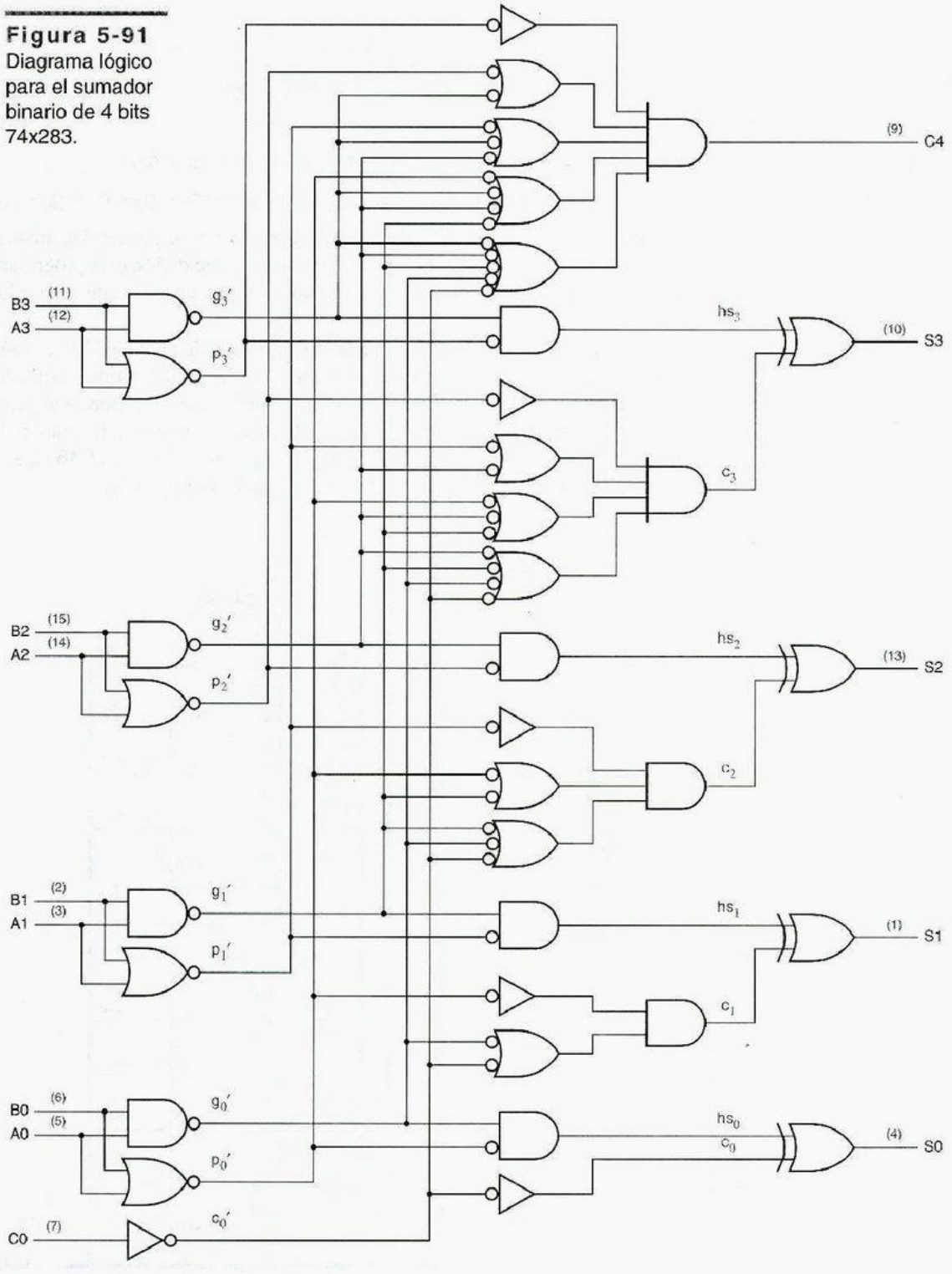


Figura 5-91
Diagrama lógico para el sumador binario de 4 bits 74x283.

$$
\begin{aligned}
c_3 &= p_2 \cdot (g_2 + c_2) \\
&= p_2 \cdot (g_2 + p_1 \cdot (g_1 + p_0) \cdot (g_1 + g_0 + c_0)) \\
&= p_2 \cdot (g_2 + p_1) \cdot (g_2 + g_1 + p_0) \cdot (g_2 + g_1 + g_0 + c_0) \\
c_4 &= p_3 \cdot (g_3 + c_3) \\
&= p_3 \cdot (g_3 + p_2 \cdot (g_2 + p_1) \cdot (g_2 + g_1 + p_0) \cdot (g_2 + g_1 + g_0 + c_0)) \\
&= p_3 \cdot (g_3 + p_2) \cdot (g_3 + g_2 + p_1) \cdot (g_3 + g_2 + g_1 + p_0) \cdot (g_3 + g_2 + g_1 + g_0 + c_0)
\end{aligned}
$$

Si usted ha seguido la derivación de estas ecuaciones y puede obtener las mismas mediante la lectura del diagrama lógico del '283, entonces ¡felicidades, está progresando rápidamente en el álgebra de conmutación! Si no, es recomendable que repase las secciones 4.1 y 4.2.

sumador iterativo en grupo

El retardo de propagación desde la entrada **C0** hasta la salida **C4** del '238 es muy breve, aproximadamente el mismo que el de dos compuertas invertidas. Como resultado, *sumadores iterativos en grupo* con más de cuatro bits pueden hacerse simplemente al poner en cascada las salidas y entradas de acarreo de los '238, como se muestra en la figura 5-92 para un sumador de 16 bits. El retardo de propagación total desde **C0** hasta **C16** en este circuito aproximadamente el mismo que el de ocho compuertas invertidas.

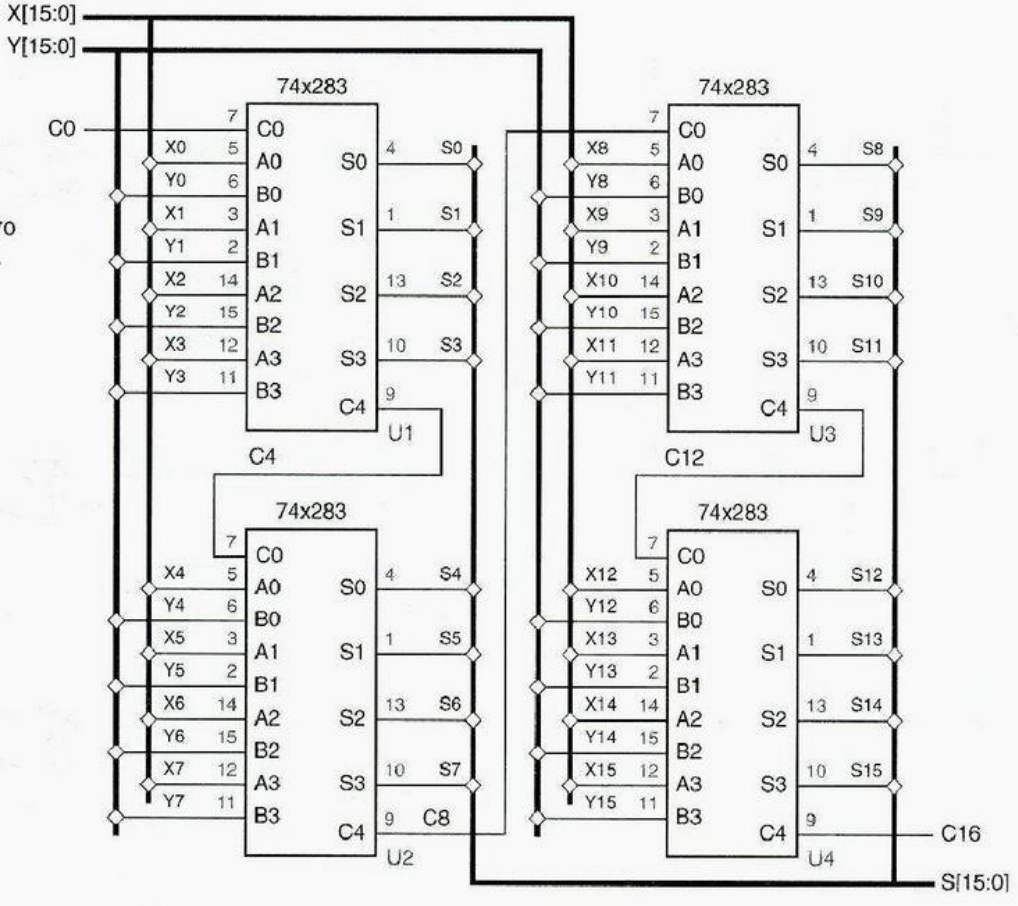


Figura 5-92
Un sumador iterativo en grupo de 16 bits.

*5.10.6 Unidades lógico-aritméticas MSI (ALU)

unidad lógica y aritmética (ALU)

Una *unidad lógico-aritmética (ALU, arithmetic and logic unit)* es un circuito combinacional que puede efectuar cualquiera de diversas operaciones lógicas y aritméticas diferentes en un par de operandos de b bits. La operación por realizarse se especifica mediante un conjunto de entradas de selección de función. Las ALU MSI típicas tienen operandos de 4 bits y de tres a cinco entradas de selección de función, permitiendo que se realicen hasta 32 diferentes funciones.

74x181

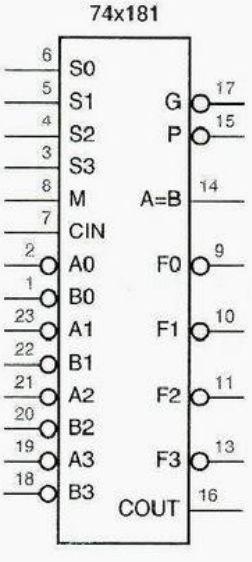
La figura 5-93 es un símbolo lógico para la ALU de 4 bits 74x181. La operación realizada por el '181 es seleccionada por M y las entradas S3-S0, como se detalla en la tabla 5-52. Nótese que los identificadores A, B y F en la tabla se refieren a las palabras de 4 bits A3-A0, B3-B0 y F3-F0; y los símbolo · y + se refieren a las operaciones lógicas AND y OR.

La entrada M del 181 selecciona entre operaciones aritméticas y lógicas. Cuando M = 1, se seleccionan las operaciones lógicas, y cada salida Fi es una función sólo de las correspondientes entradas de datos, Ai y Bi. No se propagan acarreos entre etapas y la entrada CIN es ignorada. Las salidas S3-S0 seleccionan una operación lógica particular; cualquiera de las 16 diferentes funciones lógicas combinacionales en dos variables puede ser seleccionada.

Tabla 5-52 Funciones realizadas por la ALU de 4 bits 74x181.

Entradas				*Función*	
S3	S2	S1	S0	***M = 0 (aritmética)***	***M = 1 (lógica)***
0	0	0	0	F = A menos 1 más CIN	F = A′
0	0	0	1	F = A · B menos 1 más CIN	F = A′ + B′
0	0	1	0	F = A · B′ menos 1 más CIN	F = A′ + B
0	0	1	1	F = 1111 más CIN	F = 1111
0	1	0	0	F = A más (A + B′) más CIN	F = A′ · B′
0	1	0	1	F = A · B más (A + B′) más CIN	F = B′
0	1	1	0	F = A menos B menos 1 más CIN	F = A ⊕ B′
0	1	1	1	F = A + B′ más CIN	F = A + B′
1	0	0	0	F = A más (A + B) más CIN	F = A′ · B
1	0	0	1	F = A más B más CIN	F = A ⊕ B
1	0	1	0	F = A · B′ más (A + B) más CIN	F = B
1	0	1	1	F = A + B más CIN	F = A + B
1	1	0	0	F = A más A más CIN	F = 0000
1	1	0	1	F = A · B más A más CIN	F = A · B′
1	1	1	0	F = A · B′ más A más CIN	F = A · B
1	1	1	1	F = A más CIN	F = A

Figura 5-93
Símbolo lógico para la ALU de 4 bits 74x181.



Cuando M = 0, se seleccionan las operaciones aritméticas, los acarreos se propagan entre las etapas y CIN se utiliza como una entrada de acarreo a la etapa menos significativa. Para operaciones más grandes de cuatro bits, pueden ponerse en cascada múltiples ALU '181 como el sumador iterativo en grupo de la figura 5-92, con la salida de acarreo (COUT) de cada ALU conectada a la entrada de acarreo (CIN) de la siguiente etapa más significativa. Las mismas señales de selección de función (M, S3-SO) se aplican a todos los '181 en la cascada.

Para realizar la suma de complemento a dos, empleamos S3-S0 para seleccionar la operación "A más B más CIN". La entrada CIN de la ALU menos significativa se establece normalmente en 0 durante las operaciones de suma. Para realizar la resta de complemento a dos, utilizamos S3-S0 para seleccionar la operación A menos B menos CIN. En este caso, la entrada CIN de la ALU menos significativa se establece normalmente a 1, ya que CIN actúa como el complemento del préstamo durante la resta.

El '181 proporciona otras operaciones aritméticas, tales como "A menos 1 más CIN", que son útiles en algunas aplicaciones (por ejemplo, decremento en 1). También proporciona un montón de operaciones aritméticas extrañas, tales como "A · B' más (A + B) más CIN", que casi nunca son usadas en la práctica, pero que "salen" del circuito gratis.

Advierta que las entradas de operando A3_L–A0_L y B3_L–B0_L, y las salidas de función F3_L–F0_L del '181 son activas bajas. El '181 también puede ser utilizado con entradas de operando y salidas de función activas altas. En este caso, debe construirse una versión diferente de la tabla de funciones. Cuando M = 1, las operaciones lógicas todavía son realizadas, pero para una combinación dada de entrada en S3–S0, la función obtenida es precisamente la dual de la numerada en la tabla 5-52. Cuando M = 0, se efectúan las operaciones aritméticas, pero la tabla de funciones es, una vez más, diferente. Consulte una hoja de datos del '181 para más detalles.

74x381
74x382

Otras dos ALU MSI, el *74x381* y el *74x382* mostrados en la figura 5-94, codifican sus entradas de selección de manera más compacta, y proporcionan solamente ocho diferentes pero útiles funciones, como se detalla en la tabla 5-53. La única diferencia entre el '381 y el '382 es que uno proporciona salidas de acarreo anticipado en grupo (lo que explicaremos a continuación), mientras que el otro proporciona salidas de desbordamiento y acarreo iterativo.

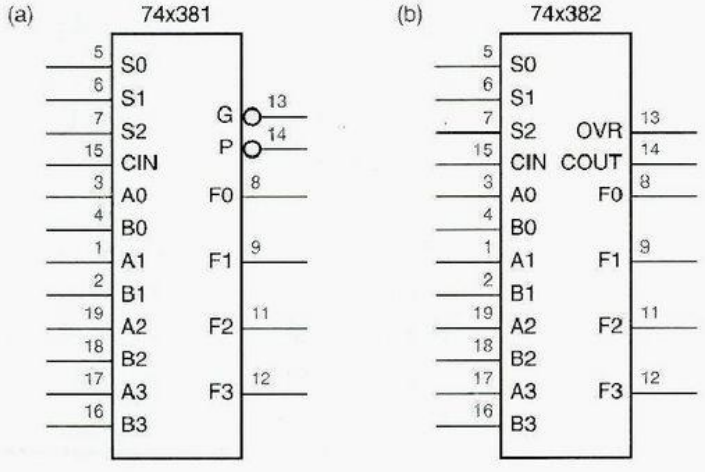


Figura 5-94
Símbolos lógicos para los ALU de 4 bits: (a) 74x381; (b) 74x382.

Tabla 5-53
Funciones realizadas por las ALU de 4 bits 74x381 y el 74x382.

Entradas S2	Entradas S1	Entradas S0	Función
0	0	0	F = 0000
0	0	1	F = B menos A menos 1 más CIN
0	1	0	F = A menos B menos 1 más CIN
0	1	1	F = A más B más CIN
1	0	0	F = A ⊕ B
1	0	1	F = A + B
1	1	0	F = A · B
1	1	1	F = 1111

*5.10.7 Acarreo de grupo anticipado

acarreo de grupo anticipado

El '181 y el '381 proporcionan salidas de *acarreo de grupo anticipado* que permiten que múltiples ALU se pongan en cascada sin acarreos iterativos entre los grupos de 4 bits. Como el 74x283, las ALU utilizan acarreo anticipado para producir acarreos internamente. Sin embargo, también proporcionan salidas G_L y P_L que son señales de acarreo anticipado para todo el grupo de 4 bits. La salida G_L es asertiva si la ALU genera un acarreo, es decir, si llega a producir un acarreo de salida (COUT = 1), exista o no un acarreo de entrada (CIN = 1):

$$G\_L = (g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0)'$$

La salida P_L es asertiva si la ALU propaga un acarreo, es decir, si llega a producir un acarreo de salida, en caso de existir un acarreo de entrada:

$$P\_L = (p_3 \cdot p_2 \cdot p_1 \cdot p_0)'$$

circuito de acarreo anticipado 74x182

Cuando las ALU se ponen en cascada, las salidas de acarreo de grupo anticipado pueden combinarse en dos niveles de lógica para producir la entrada de acarreo para cada ALU. Un *circuito de acarreo anticipado*, el *74x182* ilustrado en la figura 5-95, realiza esta operación. Las entradas del '182 son C0, la entrada de acarreo a la ALU menos significativa ("ALU 0") y G0−G3 y P0−P3, las salidas generada y propagada de las ALU 0−3. Utilizando estas entradas, el '182 produce entradas de acarreo C1−C3 para las ALU 1−3. La figura 5-96 muestra las conexiones para un ALU de 16 bits que emplea cuatro '381 y un '182.

Las ecuaciones de acarreo del '182 se obtienen al "distribuir la suma" en la ecuación básica de acarreo hacia adelante, estudiada en la sección 5.10.4:

$$\begin{aligned} c_{i+1} &= g_i + p_i \cdot c_i \\ &= (g_i + p_i) \cdot (g_i + c_i) \end{aligned}$$

Expandiendo para los primeros tres valores de i, obtenemos las siguientes ecuaciones:

$$C1 = (G0+P0) \cdot (G0+C0)$$
$$C2 = (G1+P1) \cdot (G1+G0+P0) \cdot (G1+G0+C0)$$
$$C3 = (G2+P2) \cdot (G2+G1+P1) \cdot (G2+G1+G0+P0) \cdot (G2+G1+G0+C0)$$

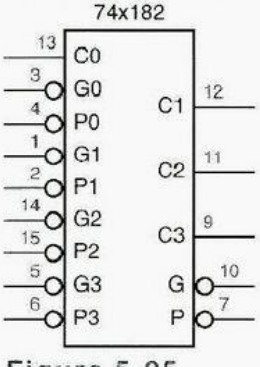
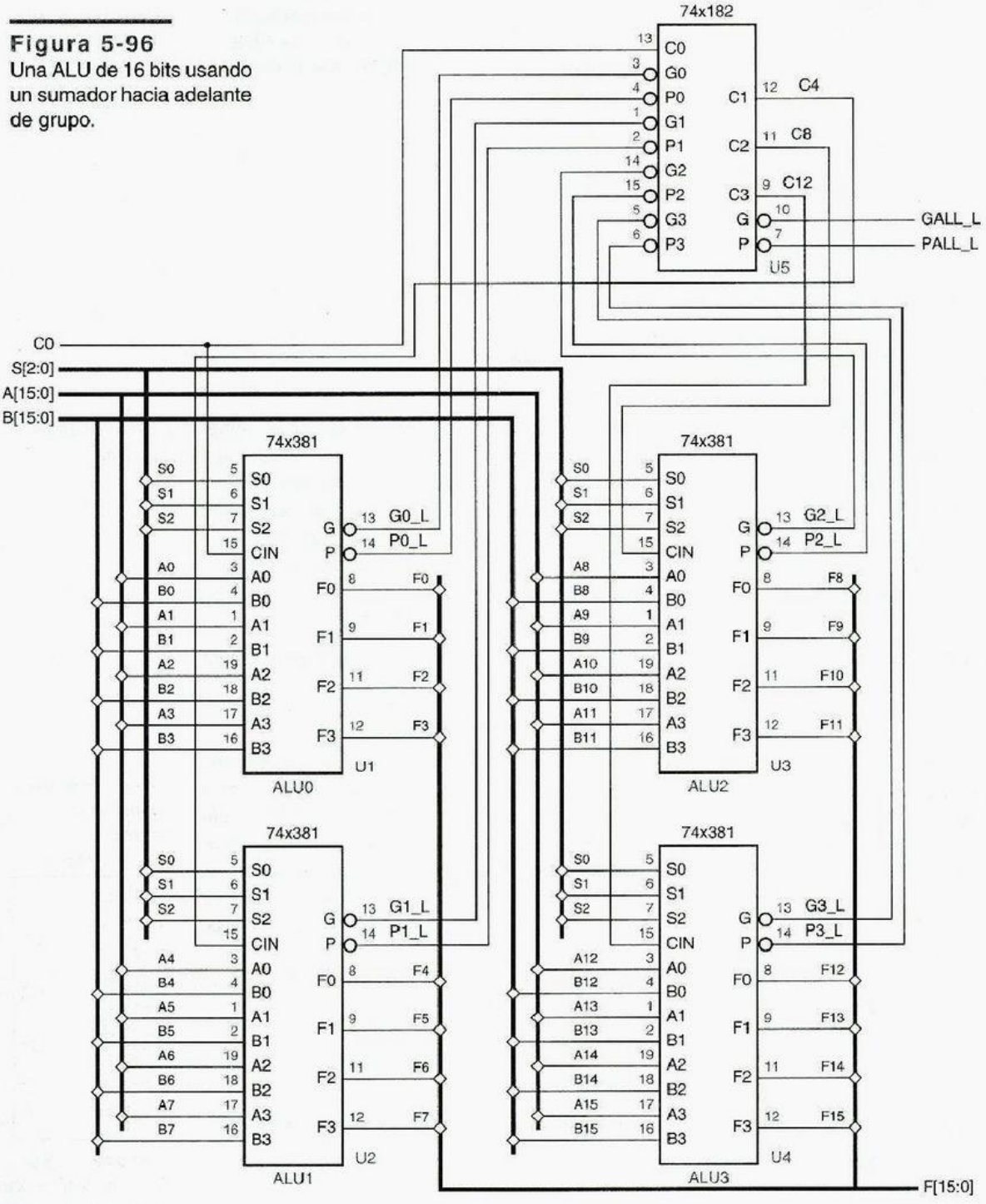


Figura 5-95
Símbolo lógico para el circuito de acarreo anticipado 74x182.

Figura 5-96
Una ALU de 16 bits usando un sumador hacia adelante de grupo.



El '182 realiza cada una de estas ecuaciones con sólo un nivel de retardo; una compuerta INVERSOR-OR AND.

Cuando más de cuatro ALU se encuentran en cascada, éstas se pueden dividir en "supergrupos", cada uno con su propio '182. Por ejemplo, un sumador de 64 bits tendría cuatro supergrupos, cada uno con cuatro ALU y un '182. Las salidas G_L y P_L de cada '182 pueden combinarse en un '182 del siguiente nivel, puesto que ellas indican si el supergrupo genera o propaga acarreos:

$$G\_L = ((G3+P3)\cdot(G3+G2+P2)\cdot(G3+G2+G1+P1)\cdot(G3+G2+G1+G0))'$$

$$P\_L = (P0\cdot P1\cdot P2\cdot P3)'$$

*5.10.8 Sumadores en ABEL y PLD

ABEL respalda los operadores de suma (+) y resta (-) que se pueden aplicar a conjuntos. Éstos son interpretados como enteros sin signo; por ejemplo, un conjunto con n bits representa un entero en el intervalo de 0 a $2^n - 1$. La resta se realiza mediante la negación del sustraendo y su posterior suma. La negación se efectúa en complemento a dos; es decir, el operando es complementado bit a bit y entonces se agrega 1.

La tabla 5-54 muestra un ejemplo de suma en ABEL. La definición del conjunto para SUM fue hecha un bit más amplia que los sumandos para acomodar el acarreo de salida del MSB; de otro modo este acarreo sería descartado. Las definiciones de conjunto para los sumandos fueron extendidas a la izquierda con un bit 0 para coincidir con el tamaño de SUM.

Aun cuando el programa sumador es extremadamente pequeño, le toma bastante tiempo compilar y generar un inmenso número de términos en el producto de sumas de dos niveles mínima. Mientras que SUM0 tiene solamente dos términos de producto, los términos subsiguientes SUMi tienen $¡5 \cdot 2^i - 4$ términos, o 636 términos para SUM7! Y el acarreo de salida (SUM8) tiene $2^{80} - 1 = 255$ términos de producto. Obviamente, los sumadores con más bits prácticamente no se pueden realizar utilizando dos niveles de lógica.

Tabla 5-54
Programa ABEL para un sumador de 8 bits.

```
module add
title 'Adder Exercise'

" Input and output pins
A7..A0, B7..B0          pin;
SUM8..SUM0              pin istype 'com';

" Set definitions
A = [0, A7..A0];
B = [0, B7..B0];
SUM = [SUM8..SUM0];

equations
SUM = A + B;

end add
```

ACARREO La salida de acarreo (SUM8) en nuestro ejemplo de sumador tiene exactamente el mismo número de términos de producto (255) que la salida "menor que" o "mayor que" de un comparador de 8 bits. Esto es menos sorprendente una vez que usted se da cuenta de que la salida de acarreo de la suma A + B es funcionalmente equivalente a la expresión $A>\overline{B}$.

directiva @CARRY

Reconociendo que los comparadores y sumadores grandes aún son necesarios en los PLD de tiempo en tiempo, ABEL proporciona una directa *@CARRY* que le dice al compilador que sintetice un sumador iterativo de grupo con *n* bits por grupo. Por ejemplo, si la instrucción "@CARRY 1;" se incluyera en la tabla 5-54, el compilador crearía ocho nuevas señales como los acarreos hacia fuera de las posiciones de bit 0 hasta 7. Las ecuaciones para SUM1 hasta SUM8 usarían estos acarreos internos, creando así esencialmente un sumador iterativo de 8 etapas con un retardo, en el peor de los casos, de ocho pasos a través del PLD.

Si se empleara la instrucción "@CARRY 2;" el compilador calcularía acarreos dos bits a la vez, creando cuatro nuevas señales para acarreos fuera de las posiciones de bits 1, 3, 5 y 7. En este caso, el número máximo de términos de producto necesarios para cualquier salida todavía es razonable, solamente 7, y la trayectoria de retardo en el peor de los casos tiene sólo cuatro pasos a través del PLD. Con tres bits por grupo (@CARRY 3;), el máximo número de términos de producto se eleva a 28, lo que es poco práctico.

Un caso especial que es frecuentemente utilizado en ABEL y PLD es la suma o resta de un 1 constante. Esta operación se utiliza en la definición de contadores, donde el estado siguiente del contador es precisamente el estado actual más 1 para un contador "hacia arriba" o menos 1 para un contador "hacia abajo". La ecuación para el bit *i* de un contador "hacia arriba" puede ser establecida simplemente en palabras: "complemente el bit *i* si el conteo está habilitado y todos los bits menores que *i* son 1". Esto requiere sólo *i* + 2 términos de producto para cualquier valor de *i*, y puede reducirse, además, a sólo un término de producto y una compuerta XOR en algunos PLD y CPLD.

*5.10.9 Sumadores en VHDL

Aunque VHDL tiene operadores integrados de suma (+) y resta (-), solamente trabajan con tipos entero, real y físico. Específicamente *no* trabajan con tipos BIT_VECTOR del tipo IEEE estándar STD_LOGIC_VECTOR. En cambio, los paquetes estándar definen estos operadores.

Como explicamos en la sección 5.9.6, el paquete IEEE_std_logic_arith define dos nuevos tipos de arreglo, SIGNED y UNSIGNED, y un conjunto de funciones de comparación para operandos del tipo INTEGER, SIGNED, o UNSIGNED. El paquete define también las operaciones de suma y resta para las mismas clases de operandos, además de STD_LOGIC y STD_ULOGIC, para operandos de 1 bit.

El gran número de funciones de suma y resta sobrepuestas puede hacer menos obvio el tipo que tendrá el resultado de suma o resta. Normalmente, si uno de los operandos es de tipo SIGNED, el resultado es SIGNED; si no es UNSIGNED. Sin embargo, si el resultado es asignado a una señal o variable de tipo STD_LOGIC_VECTOR, entonces el resultado SIGNED o UNSIGNED se convierte en ese tipo. La longitud de cualquier

Tabla 5-55
Programa VHDL para sumar y restar enteros de 8 bits de diversos tipos.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity vadd is
    port (
        A, B: in UNSIGNED (7 downto 0);
        C: in SIGNED (7 downto 0);
        D: in STD_LOGIC_VECTOR (7 downto 0);
        S: out UNSIGNED (8 downto 0);
        T: out SIGNED (8 downto 0);
        U: out SIGNED (7 downto 0);
        V: out STD_LOGIC_VECTOR (8 downto 0)
    );
end vadd;

architecture vadd_arch of vadd is
  begin
    S <= ('0' & A) + ('0' & B);
    T <= A + C;
    U <= C + SIGNED(D);
    V <= C - UNSIGNED(D);
end vadd_arch;
```

resultado es normalmente la del operando más extenso. Sin embargo, cuando un operando UNSIGNED se combina con un SIGNED o un INTEGER, su longitud se incrementa por 1 para acomodar un bit de signo de 0, y entonces la longitud del resultado es determinada.

Incorporando estas consideraciones, el programa VHDL en la tabla 5-55 muestra sumas de 8 bits para varios operandos y tipos de resultados. El primer resultado, S, se declara de 9 bits de longitud suponiendo que el diseñador se encuentra interesado en el acarreo de la suma de 8 bits de los operandos UNSIGNED A y B. El operador de concatenación & se utiliza para extender A y B, de modo que la función de suma regresará el bit de acarreo en el MSB del resultado.

El resultado siguiente, T, también es de 9 bits de longitud, puesto que la función de suma extiende el operando A UNSIGNED cuando se combina con el operando C SIGNED. En la tercera suma, un STD_LOGIC_VECTOR D de 8 bits es convertido en tipo a SIGNED y combinado con C para obtener un resultado U SIGNED de 8 bits. En la última instrucción, D se convierte en UNSIGNED, automáticamente extendida por un bit, y restada de C para producir un resultado V de 9 bits.

Puesto que la suma y la resta son bastante costosas en términos del número de compuertas requeridas, muchos motores de síntesis VHDL intentarán reutilizar los bloques de sumador siempre que sea posible. Por ejemplo, la tabla 5-56 es un programa VHDL que incluye dos sumas diferentes. Más que construir dos sumadores y seleccionar la salida de uno con un multiplexor, el motor de síntesis puede construir sólo un sumador

Tabla 5-56
Programa VHDL que permite sumador compartido.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity vaddshr is
    port (
        A, B, C, D: in SIGNED (7 downto 0);
        SEL: in STD_LOGIC;
        S: out SIGNED (7 downto 0)
    );
end vaddshr;

architecture vaddshr_arch of vaddshr is
begin
    S <= A + B when SEL = '1' else C + D;
end vaddshr_arch;
```

y seleccionar sus entradas con la ayuda de multiplexores, creando potencialmente un circuito completo más pequeño.

*5.11 Multiplicadores combinacionales

*5.11.1 Estructuras de multiplicador combinacional

En la sección 2.8 esbozamos un algoritmo que utiliza n corrimientos y sumas para multiplicar números binarios de n bits. Aunque el algoritmo de suma y acarreo de salida emula la forma en que hacemos la multiplicación de números decimales con papel y lápiz, no hay nada inherentemente "secuencial" o "dependiente del tiempo" acerca de la multiplicación. Es decir, dadas dos palabras de entrada de n bits X y Y, es posible escribir una tabla de verdad que exprese el producto $P = X \cdot Y$ de $2n$ bits como una función *combinacional* de X y Y. Un *multiplicador combinacional* es un circuito lógico con una tabla de verdad de esta clase.

miltiplicador combinacional

La mayor parte de los métodos de la multiplicación combinacional están basados en el algoritmo de suma y acarreo de salida de papel y lápiz. La figura 5-97 ilustra la idea básica para un multiplicador de 8 X 8 para dos enteros sin signo, el multiplicando $X = x_7x_6x_5x_4x_3x_2x_1x_0$ y el multiplicador $Y = y_7y_6y_5y_4y_3y_2y_1y_0$. Llamamos a cada renglón un

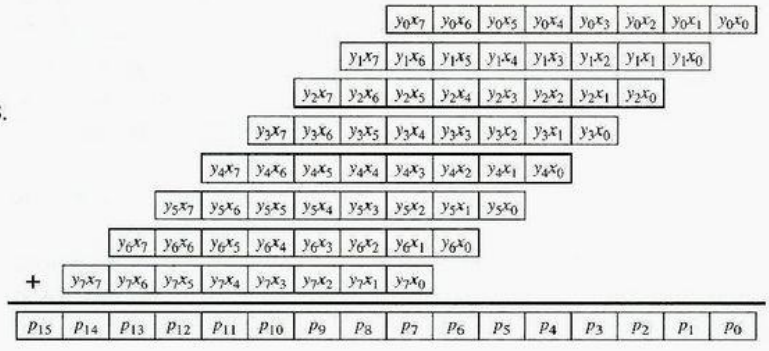


Figura 5-97
Productos parciales en un multiplicador de 8 × 8.

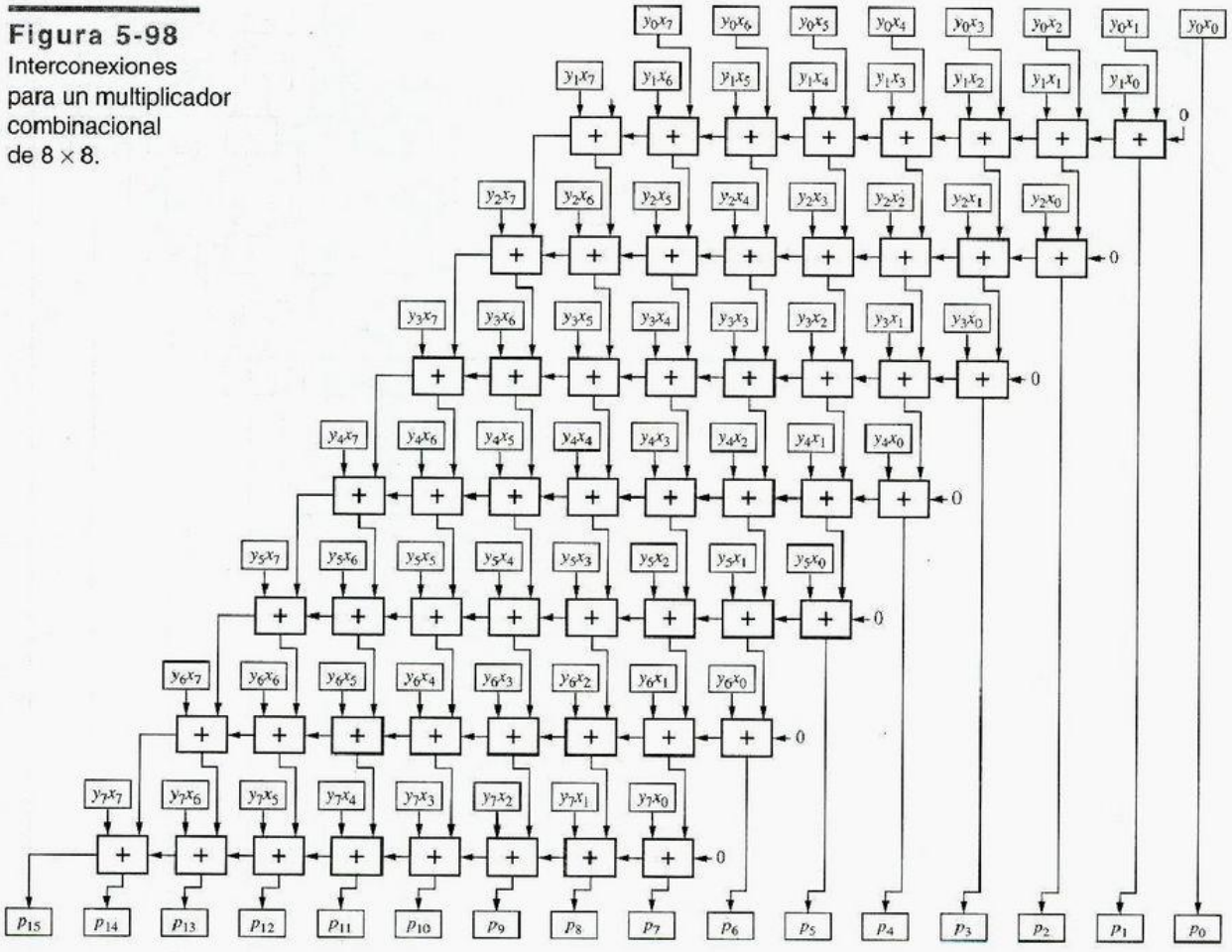


Figura 5-98
Interconexiones para un multiplicador combinacional de 8×8.

componente de producto, un multiplicando con acarreo de salida que es multiplicado por 0 o por 1 dependiendo del correspondiente bit multiplicador. Cada pequeño cuadro representa un bit componente de producto $y_i x_j$, el AND lógico del bit multiplicador y_i y el bit multiplicando x_j. El producto $P = p_{15}p_{14}\cdots p_2p_1p_0$ tiene 16 bits y se obtiene al sumar en conjunto todos los componentes de producto.

componente de producto

La figura 5-98 muestra una manera de sumar los componentes de producto. Aquí, los bits de componente de producto han sido extendidos para hacer espacio, y cada cuadro "+" es un equivalente de sumador completo de la figura 5-86(c) de la página 431. Los acarreos en cada renglón de sumadores completos están conectados para hacer un sumador iterativo de 8 bits. De este modo, el primer sumador iterativo combina los primeros dos componentes de producto para producir el primer producto parcial, como se define en la sección 2.8. Los sumadores subsiguientes combinan cada producto parcial con el siguiente componente de producto.

Es interesante estudiar el retardo de propagación del circuito en la figura 5-98. En el peor de los casos, las entradas para el sumador menos significativo (y_0x_1 y y_1x_0) pueden afectar el MSB del producto (p_{15}). Si suponemos por simplicidad que los retardos de

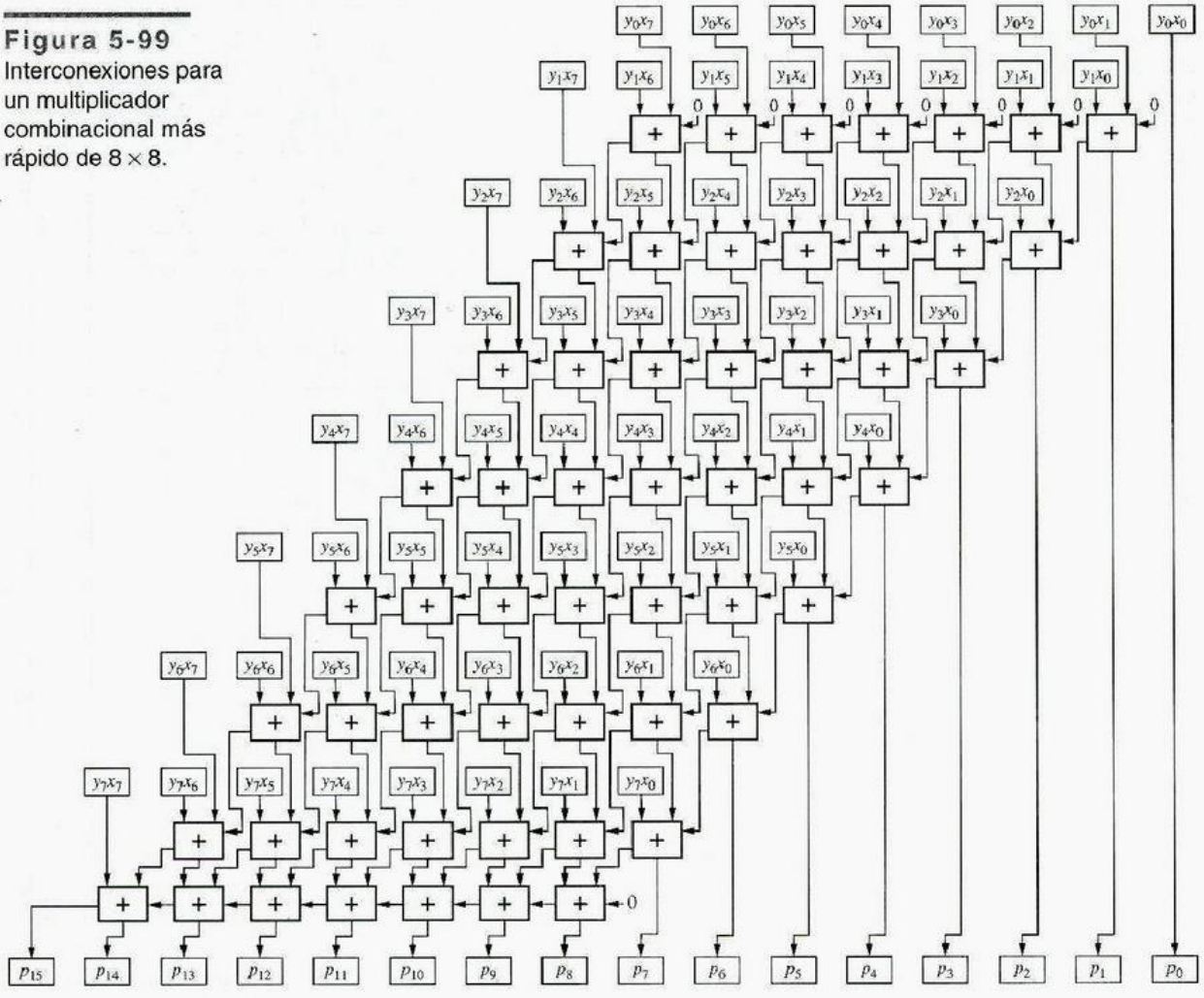


Figura 5-99
Interconexiones para un multiplicador combinacional más rápido de 8×8.

cualquier entrada hacia cualquier salida de un sumador completo son iguales, digamos t_{pd}, entonces la trayectoria en el peor de los casos pasa a través de 20 sumadores y su retardo es de $20t_{pd}$. Si los retardos son diferentes, entonces la respuesta depende de los retardos relativos; véase el ejercicio 5.83.

multiplicador secuencial

Los *multiplicadores secuenciales* utilizan un sumador simple y un registro para acumular los productos parciales. El registro de producto parcial es inicializado al primer componente de producto, y para una multiplicación de $n \times n$ bits, se toman $n - 1$ pasos y el sumador se emplea $n - 1$ veces, una vez por cada uno de los $n - 1$ componentes de producto restantes por ser sumados al registro del producto parcial.

suma de acarreo almacenado

Algunos multiplicadores secuenciales utilizan un truco conocido como *suma de acarreo almacenado* para acelerar la multiplicación. La idea es romper la cadena de acarreo del sumador iterativo para acortar el retardo de cada suma. Esto se hace al aplicar la salida del acarreo del bit i durante el paso j a la entrada de acarreo para el bit $i + 1$ durante el paso *siguiente*, $j + 1$. Después de que se suma el último componente de producto, se necesita

un paso más en el que los acarreos se conectan de la manera habitual y se permite el ciclo iterativo desde el último hasta el bit más significativo.

El equivalente combinacional de un multiplicador de 8 × 8 utilizando suma de acarreo almacenado se muestra en la figura 5-99. Nótese que el acarreo de salida de cada sumador completo en los primeros siete renglones está conectado a una entrada de un sumador *debajo* de él. Los acarreos en el octavo renglón de los sumadores completos están conectados para crear un sumador iterativo convencional. Aunque este sumador utiliza exactamente la misma cantidad de lógica que el anterior (64 compuertas AND de 2 entradas y 56 sumadores completos), su retardo de propagación es sustancialmente más corto. Su trayectoria de retardo en el peor de los casos pasa a través de solamente 14 sumadores completos. El retardo puede ser mejorado aún más utilizando un sumador de acarreo anticipado para el último renglón.

La estructura regular de multiplicadores combinacionales los hace ideales para la realización VLSI y ASIC. La importancia de la multiplicación rápida en los microprocesadores, video digital y muchas otras aplicaciones ha llevado a muchos a estudiar y a desarrollar estructuras y circuitos aún mejores para los multiplicadores combinacionales; consulte las Referencias.

*5.11.2 Multiplicación en ABEL y PLD

ABEL proporciona un operador de multiplicación *, pero puede ser empleado solamente con señales, números o constantes especiales individuales, no con conjuntos. De este modo, ABEL no puede sintetizar un circuito multiplicador a partir de una ecuación simple como "P = X*Y."

Con todo, se puede utilizar ABEL para especificar un multiplicador combinacional si se le divide en varias partes más pequeñas. Por ejemplo, la tabla 5-57 muestra el diseño de un multiplicador de 4×4 sin signo, cuya estructura general es igual la de la figura 5-97, página 446. Las expresiones se utilizan para definir los cuatro componentes de producto PC1, PC2, PC3, y PC4, los cuales se suman entonces en la sección de equations del programa. Esto no genera un arreglo de sumadores completos como en la figura 5-98 o en la 5-99. En vez de ello, el compilador ABEL truncará obedientemente la ecuación

```
module mul4x4
title '4x4 Combinational Multiplier'

X3..X0, Y3..Y0 pin;   " multiplicand, multiplier
P7..P0         pin istype 'com';     " product

P = [P7..P0];
PC1 = Y0 & [0, 0, 0, 0,X3,X2,X1,X0];
PC2 = Y1 & [0, 0, 0,X3,X2,X1,X0, 0];
PC3 = Y2 & [0, 0,X3,X2,X1,X0, 0, 0];
PC4 = Y3 & [0,X3,X2,X1,X0, 0, 0, 0];

equations
P = PC1 + PC2 + PC3 + PC4;

end mul4x4
```

Tabla 5-57
Programa ABEL para un multiplicador combinacional de 4×4.

de suma para producir una suma mínima para cada uno de los ocho bits de salida de producto. De manera sorprendente, la salida en el peor de los casos, P4, tiene solamente 36 términos de producto, un poco alto pero ciertamente realizable en dos pasos a través de un PLD.

*5.11.3 Multiplicación en VHDL

VHDL es lo suficientemente rico para expresar la multiplicación de muy diversas maneras; reservaremos las mejores para el final.

La tabla 5-58 es un programa VHDL de comportamiento que imita la estructura del multiplicador de la figura 5-99. A fin de representar las señales internas en la figura, el programa define un nuevo tipo de datos, `array8x8`, el cual es un arreglo bidimensional de `STD_LOGIC` (recuerde que `STD_LOGIC_VECTOR` es un arreglo unidimensional de `STD_LOGIC`). La variable `PC` se declara como un arreglo de esta clase para mantener los bits del componente de producto, y las variables `PCS` y `PCC` son arreglos semejantes para mantener las salidas de suma y acarreo del arreglo principal de sumadores completos. Los arreglos unidimensionales `RAS` y `RAC` mantienen las salidas de suma y acarreo del sumador iterativo. La figura 5-100 muestra la nomenclatura de variables y el esquema de numeración. Las variables enteras `i` y `j` se utilizan como índices de ciclo para los renglones y las columnas, respectivamente.

El programa intenta ilustrar las compuertas lógicas que se utilizarían en una realización fiel de la figura 5-99, aun cuando un sintetizador podría legítimamente crear una estructura bastante diferente de este programa de comportamiento. Si se desea controlar la estructura, entonces se debe emplear VHDL estructural, como veremos más adelante.

En el programa, la primera instrucción anidada `for` realiza 64 operaciones AND para obtener los bits de componente de producto. El siguiente ciclo `for` inicializa las condiciones de frontera en la parte superior del multiplicador, utilizando la noción de los

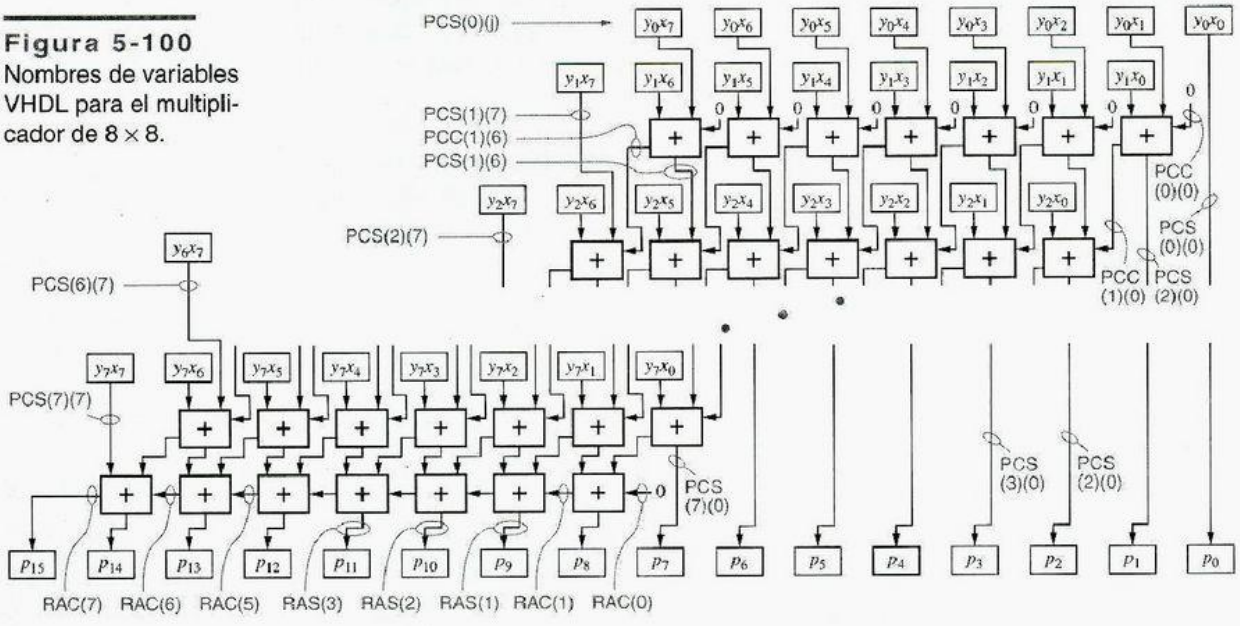


Figura 5-100
Nombres de variables VHDL para el multiplicador de 8 × 8.

Tabla 5-58 Programa VHDL de comportamiento para un multiplicador combinacional de 8×8.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity vmul8x8p is
    port ( X: in STD_LOGIC_VECTOR (7 downto 0);
           Y: in STD_LOGIC_VECTOR (7 downto 0);
           P: out STD_LOGIC_VECTOR (15 downto 0) );
end vmul8x8p;

architecture vmul8x8p_arch of vmul8x8p is
function MAJ (I1, I2, I3: STD_LOGIC) return STD_LOGIC is
  begin
    return ((I1 and I2) or (I1 and I3) or (I2 and I3));
  end MAJ;
begin
process (X, Y)
type array8x8 is array (0 to 7) of STD_LOGIC_VECTOR (7 downto 0);
variable PC: array8x8;      -- bits de componente de producto
variable PCS: array8x8;     -- bits de suma del sumador completo
variable PCC: array8x8;     -- bits de salida de acarreo del sumador completo
variable RAS, RAC: STD_LOGIC_VECTOR (7 downto 0); -- suma del sumador en rizo
  begin                                           --   y bits de acarreo
    for i in 0 to 7 loop for j in 0 to 7 loop
        PC(i)(j) := Y(i) and X(j); -- cálculo de bits de componente de producto
    end loop;  end loop;
    for j in 0 to 7 loop
      PCS(0)(j) := PC(0)(j);  -- inicializar primer renglón "virtual"
      PCC(0)(j) := '0';       --   sumadores (no mostrados en la figura)
    end loop;
    for i in 1 to 7 loop      -- hacer todos los sumadores completos excepto el
-- último renglón
      for j in 0 to 6 loop
        PCS(i)(j) := PC(i)(j) xor PCS(i-1)(j+1) xor PCC(i-1)(j);
        PCC(i)(j) := MAJ(PC(i)(j), PCS(i-1)(j+1), PCC(i-1)(j));
        PCS(i)(7) := PC(i)(7); -- señal de suma del sumador "virtual" que está
                               en el extremo izquierdo
      end loop;
    end loop;
    RAC(0) := '0';
    for i in 0 to 6 loop  -- sumador en rizo final
      RAS(i) := PCS(7)(i+1) xor PCC(7)(i) xor RAC(i);
      RAC(i+1) := MAJ(PCS(7)(i+1), PCC(7)(i), RAC(i));
    end loop;
    for i in 0 to 7 loop
     P(i) <= PCS(i)(0);  -- primeros 8 bits de producto de las sumas del sumador
-- completo
    end loop;
    for i in 8 to 14 loop
      P(i) <= RAS(i-8);   -- siguientes 7 bits de las sumas del sumador en rizo
    end loop;
    P(15) <= RAC(7);      -- último bit del acarreo del sumador en rizo
  end process;
end vmul8x8p_arch;
```

Tabla 5-59 Arquitectura VHDL estructural para un multiplicador combinacional de 8×8.

```
architecture vmul8x8s_arch of vmul8x8s is
component AND2
  port( I0, I1: in STD_LOGIC;
        O: out STD_LOGIC );
end component;
component XOR3
  port( I0, I1, I2: in STD_LOGIC;
        O: out STD_LOGIC );
end component;
component MAJ   -- función mayoría, O = I0*I1 + I0*I2 + I1*I2
  port( I0, I1, I2: in STD_LOGIC;
        O: out STD_LOGIC );
end component;

type array8x8 is array (0 to 7) of STD_LOGIC_VECTOR (7 downto 0);
signal PC: array8x8;      -- bits de componente de producto
signal PCS: array8x8;     -- bits de suma de sumador completo
signal PCC: array8x8;     -- bits de salida de acarreo de sumador completo
signal RAS, RAC: STD_LOGIC_VECTOR (7 downto 0); -- suma, acarreo
begin
  g1: for i in 0 to 7 generate    -- bits de componente de producto
    g2: for j in 0 to 7 generate
        U1: AND2 port map (Y(i), X(j), PC(i)(j));
    end generate;
  end generate;
  g3: for j in 0 to 7 generate
    PCS(0)(j) <= PC(0)(j);  -- inicializar sumadores "virtuales"
                            -- de primer renglón
    PCC(0)(j) <= '0';
  end generate;
  g4: for i in 1 to 7 generate  -- efectuar los sumadores completos,
                                -- a excepción del último renglón
   g5: for j in 0 to 6 generate
     U2: XOR3 port map (PC(i)(j),PCS(i-1)(j+1),PCC(i-1)(j),PCS(i)(j));
     U3: MAJ  port map (PC(i)(j),PCS(i-1)(j+1),PCC(i-1)(j),PCC(i)(j));
     PCS(i)(7) <= PC(i)(7); -- salida de suma del sumador "virtual"
                            -- que está en el extremo izquierdo
   end generate;
  end generate;
  RAC(0) <= '0';
  g6: for i in 0 to 6 generate  -- sumador iterativo final
    U7: XOR3 port map (PCS(7)(i+1), PCC(7)(i), RAC(i), RAS(i));
    U3: MAJ  port map (PCS(7)(i+1), PCC(7)(i), RAC(i), RAC(1+i));
  end generate;
  g7: for i in 0 to 7 generate
    P(i) <= PCS(i)(0); -- obtener los primeros 8 bits de producto
                       -- de las sumas del sumador completo
  end generate;
  g8: for i in 8 to 14 generate
    P(i) <= RAS(i-8);  -- obtener los siguientes 7 bits e las sumas del
                       -- sumador en rizo
  end generate;
  P(15) <= RAC(7);     -- obtener el último bit del acarreo del sumador
                       -- en rizo
end vmul8x8_arch;
```

sumadores completos "virtuales" del renglón 0, no mostrados en la figura, cuyas salidas de suma igualan el primer renglón de bits de PC, mientras que su acarreo de salida es 0. El tercer ciclo anidado `for` corresponde al arreglo principal de sumadores en la figura 5-99, todos excepto el último renglón, el cual es manipulado por el cuarto ciclo `for`. Los últimos dos ciclos `for` asignan las salidas de sumador apropiadas a las señales de salida del multiplicador.

El programa de la tabla 5-58 puede ser modificado para emplear VHDL estructural como se muestra en la tabla 5-59. Este enfoque proporciona un control completo al diseñador sobre la estructura del circuito que es sintetizado, como puede desearse en una realización ASIC. El programa supone que las arquitecturas para `AND2`, `XOR3` y `MAJ3` han sido definidas en otro sitio, por ejemplo, en una biblioteca ASIC.

instrucción `generate`

Este programa hace buen uso de la *instrucción* `generate` para crear los arreglos de componentes utilizados en el multiplicador. La instrucción `generate` debe tener una etiqueta y, de modo similar para una instrucción de iteración `for-loop`, especifica un esquema de iteración para controlar la repetición de las instrucciones encerradas dentro del mismo. Dentro del `for-generate`, las instrucciones encerradas pueden incluir cualquier instrucción concurrente, instrucciones `IF-THEN-ELSE`, y niveles adicionales de construcciones de ciclos. En ocasiones, las instrucciones `generate` se combinan con `IF-THEN-ELSE` para producir una especie de capacidad de compilación condicional.

Bien, dijimos que habíamos reservado lo mejor para el final, y aquí lo tenemos. La biblioteca IEEE `std_logic_arith` que presentamos en la sección 5.9.6 define funciones de multiplicación para tipos `SIGNED` y `UNSIGNED`, y sobrepone estas funciones sobre el operador "*". De esta forma, el programa de la tabla 5-60 puede multiplicar números sin signo con una simple instrucción de asignación de una línea.

Dentro de la biblioteca IEEE `std_logic_arith`, la función de multiplicación se define de manera "conductista", esto es, de comportamiento, con la ayuda del algoritmo de suma y corrimiento. Podríamos haber mostrado este enfoque desde el principio de esta subsección, pero entonces usted no habría leído todo lo demás, ¿o sí?

Tabla 5-60
Programa VHDL verdaderamente de comportamiento para un multiplicador combinacional de 8×8.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity vmul8x8i is
    port (
        X: in UNSIGNED (7 downto 0);
        Y: in UNSIGNED (7 downto 0);
        P: out UNSIGNED (15 downto 0)
    );
end vmul8x8i;

architecture vmul8x8i_arch of vmul8x8i is
begin
  P <= X * Y;
end vmul8x8i_arch;
```

SEÑALES VERSUS VARIABLES

Las variables se utilizan más que las señales en el proceso de la tabla 5-58, para hacer que la simulación se ejecute más rápido. Las variables son más rápidas porque el simulador sigue la pista de sus valores solamente cuando el proceso se encuentra en ejecución. Debido a que los valores de las variables se asignan en forma secuencial, el proceso en la tabla 5-58 se escribe con mucho cuidado para calcular valores en el orden adecuado. Es decir, una variable no se puede utilizar hasta que se haya asignado un valor a ella.

Las señales, por otra parte, tienen un valor en todo momento. Cuando el valor de una señal se modifica durante un proceso, el simulador programa un evento futuro en su lista de eventos para el cambio de valor. Si la señal aparece en el lado derecho de una instrucción de asignación en el proceso, entonces se debe incluir la señal en la lista de sensibilidad del proceso. Si un valor de señal cambia, entonces el proceso se ejecutará de nueva cuenta, y se mantendrá repitiéndose hasta que todas las señales en la lista de sensibilidad estén estables.

En la tabla 5-58, si usted quiere observar los valores internos de temporización durante la simulación, podría cambiar todas las variables (excepto `i` y `j`) a señales, e incluirlas en la lista de sensibilidad. Para hacer el programa sintácticamente correcto, también tendría que mover las declaraciones `type` y `signal` justamente después de la instrucción `architecture` y cambiar todas las asignaciónes ":=" a "<=".

La simulación se ejecutará con mayor lentitud después de haber realizado los cambios antes mencionados. Como discutimos en conexión con la figura 5-99 de la página 448, la trayectoria de propagación de la señal en el peor de los casos a través del arreglo del sumador pasa a través de 14 sumadores. Éste es también el número de ciclos de simulación que deben ocurrir para que se estabilicen las señales en el programa VHDL modificado, más un ciclo adicional para el simulador, para indicar que se han estabilizado.

La elección de señales contra variables afecta la velocidad de simulación, pero en la mayor parte de las máquinas de síntesis VHDL no se afectan los resultados de la síntesis.

SOBRE EL UMBRAL DE UN SUEÑO

Una "función de mayoría" de triple entrada, `MAJ`, se define al principio de la tabla 5-58 y se utiliza en lo subsiguiente para calcular salidas de acarreo. Una *función de mayoría* de n entradas produce una salida 1, si la mayoría de las entradas son 1; dos de tres, en el caso de una función de mayoría de 3 entradas. (Si n es par, $n/2 + 1$ entradas deber ser 1).

Hace 30 años, existía un sustancial interés académico en una clase más general de *funciones de umbral* de n entradas, las cuales producen un 1, si k o más de sus entradas son 1. Además de proporcionar trabajo de tiempo completo a los gurús teóricos de la lógica, las funciones de umbral podrían realizar muchas funciones lógicas con un número más pequeño de elementos que una realización convencional AND/OR. Por ejemplo, una función acarreo de sumador requiere tres compuertas AND y una COLECTOR OR, pero sólo una compuerta de umbral de tres entradas.

Por desgracia, nunca surgió una tecnología económica para las compuertas de umbral y éstas permanecen, por ahora, como una curiosidad académica.

SÍNTESIS DE DISEÑOS DE COMPORTAMIENTO

Usted probablemente habrá escuchado que los compiladores para lenguajes de programación de alto nivel, como C, por lo regular generan mejor código que las personas que escriben lenguaje ensamblador. La mayoría de los diseñadores digitales esperan que los compiladores para los HDL que se basan en el comportamiento algún día darán mejores resultados que para VHDL, esquemático o estructural. El uso de mejores compiladores no provocará que los diseñadores se queden sin trabajo, simplemente permitirá que enfrenten diseños más grandes.

Todavía no hemos llegado a ese momento. Sin embargo, las máquinas de síntesis más avanzadas incluyen algunas optimizaciones agradables para las estructuras de comportamiento que se utilizan con mayor frecuencia. Por ejemplo, tengo que admitir que la máquina de síntesis de FPGA que he utilizado para verificar los programas VHDL en esta subsección, generaron con igual rapidez un multiplicador a partir de la tabla 5-60 ¡que a partir de las arquitecturas más detalladas!

Referencias

Los diseñadores digitales que quieran mejorar sus códigos escritos deberían comenzar por leer el clásico *Elements of Style,* tercera edición, de William Strunk, Jr. y E. B. White (Allyn & Bacon, 1979). Probablemente la guía más económica y concisa y, a pesar de ello, sumamente útil para la escritura técnica es *The Elements of Technical Writing,* por Gary Blake y Robert W. Bly (Macmillan, 1993). Para un tratamiento más enciclopédico, véase *Handbook of Technical Writing,* quinta edición, por Brusaw, Alred y Oliu (St. Martin's Press, 1997).

El estándar ANSI/IEEE para símbolos lógicos es Std 91-1984, *IEEE Standard Graphic Symbols for Logic Functions.* Otro estándar de interés para diseñadores lógicos es el ANSI/IEEE 991-1986, *Logic Circuit Diagrams.* Estos dos estándares y otros diez, incluyendo los símbolos estándar y conectores de señal, se pueden encontrar en una obra de referencia práctica, *Electrical and Electronics Graphic and Letter Symbols and Reference Designations Standards Collection,* publicada por el IEEE en 1996 (`www.ieee.org`).

Los dispositivos lógicos reales se describen en hojas de especificaciones técnicas y manuales técnicos que publican los fabricantes. Las ediciones actualizadas de los manuales de datos se publican con cierta periodicidad, pero recientemente la tendencia ha sido minimizar o eliminar las ediciones en papel y, en su lugar, publicar información actualizada en la red de internet (la Web). Dos de los principales proveedores con los sitios más detallados son Texas Instruments (`www.ti.com`) y Motorola (`www.mot.com`).

Para una familia lógica dada, como la 74ALS, todos los fabricantes numeran generalmente especificaciones equivalentes, de manera que pueda conseguirlo con sólo un manual de datos técnicos por familia. Algunas especificaciones, en especial las de temporización, pueden variar un poco entre fabricantes, de modo que cuando la temporización está muy ajustada, es mejor verificar un par de diferentes fuentes y emplear la temporización del peor de los casos. Eso es *mucho* más fácil que convencer a su departamento de fabricación para que solamente adquiera un componente con un proveedor determinado.

Los primeros dispositivos PAL fueron inventados en Monolithic Memories, Inc. (MMI) en 1978, por John Birkner y H. T. Chua. Los inventores consiguieron la patente de Estados Unidos número 4,124,899 para su invento, ¡y MMI los recompensó comprándoles unos automóviles nuevos marcas Porsche y Mercedes, respectivamente! Viendo el valor de esta tecnología (los dispositivos PAL, no los automóviles), Advanced Micro Devices (AMD) adquirió MMI a principios de la década de 1980 y se convirtió en un desarrollador y proveedor principal de los nuevos PLD y CPLD. En 1997, AMD transfirió sus operaciones de PLD a una subsidiaria, Vantis Corporation, la cual fue vendida en 1999 al antiguo competidor Lattice Semiconductor.

Algunos de los mejores recursos para el aprendizaje del diseño basado en PLD son proporcionados por los fabricantes de los PLD. Xilinx Corporation, que comenzó en el negocio de FPGA, también hace CPLD, y publica un extenso *Xilinx Data Book* en rústica (San José, CA 95124, 1999) y en la Web (`www.xilinx.com`). De manera similar, los inventores de GAL, Lattice Semiconductor tienen un detallado *Lattice Data Book* (Hillsboro, OR 97124, 1999) y un sitio Web (`www.latticesemi.com`).

Un análisis más detallado del funcionamiento interno de los dispositivos LSI y VLSI, incluyendo PLD, ROM y RAM, puede hallarse en textos de electrónica tales como *Microelectronics,* segunda edición, de J. Millman y A. Grabel (McGraw-Hill, 1987) y *VLSI Engineering de* Thomas E. Dillinger (Prentice Hall, 1988).

Por el lado técnico del diseño digital, multitud de obras de texto estudian los principios del diseño digital, pero solamente unos cuantos tratan los aspectos prácticos del diseño. Un excelente y breve libro enfocado a las prácticas del diseño digital es *The Well-Tempered Digital Design, de* Robert B. Seidensticker (Addison-Wesley, 1986). Contiene cientos de "proverbios" fácilmente accesibles de diseño digital, en áreas que van desde la filosofía de diseño de alto nivel hasta la capacidad de fabricación.

Problemas propuestos

5.1 Proporcione tres ejemplos de circuitos lógicos combinacionales que requieran *miles y miles de millones* de renglones para describir una tabla de verdad. Para cada circuito, describa sus entradas y salida(s) e indique exactamente cuántos renglones contiene la tabla de verdad; no necesita escribirla. (*Sugerencia:* puede hallar tales circuitos justo en este capítulo).

5.2 Dibuje el símbolo equivalente de DeMorgan para una compuerta NAND 74x30 de 8 entradas.

5.3 Dibuje el símbolo equivalente de DeMorgan para una compuerta NOR 74x27 de 3 entradas.

5.4 ¿Qué hay de erróneo en el nombre de la señal "READY"?

5.5 Puede llegarle a molestar tener que seguir la pista de los niveles activos de todas las señales en un circuito lógico. ¿Por qué no emplear solamente chips no inversores, de modo que todas las señales sean activas altas?

5.6 Verdadero o falso: en el diseño lógico en burbuja, las salidas con una burbuja pueden conectarse solamente a entradas con una burbuja.

5.7 Un sistema de comunicación digital está siendo diseñado con 12 puertos de red idénticos. ¿Qué tipo de estructura esquemática es probablemente la más apropiada para el diseño?

5.8 Determine el retardo de propagación máximo exacto desde IN hasta OUT del circuito de la figura X5.8, tanto para transiciones BAJO a ALTO como para ALTO a BAJO, utilizando la información de temporización dada en la tabla 5-2. Repita, con la ayuda de un número de retardo, simple en el peor de los casos, para cada compuerta, y compare sus resultados. Haga un comentario acerca de estos últimos.

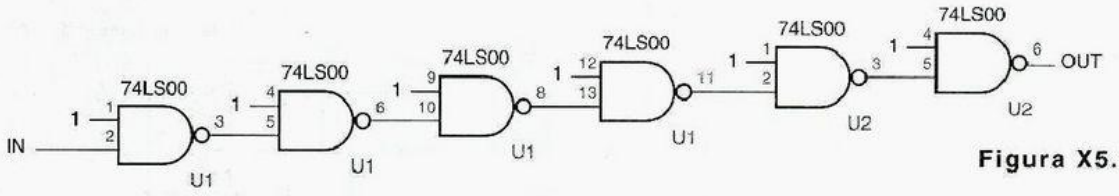


Figura X5.8

5.9 Repita el problema 5.8, sustituyendo los 74HCT00 por los 74LS00.

5.10 Repita el problema 5.8, sustituyendo los 74LS08 por los 74LS00.

5.11 Repita el problema 5.8, sustituyendo los 74AHCT02 por los 74LS00, con la ayuda de las entradas 0 constantes en lugar de las 1 constantes, y empleando temporización típica en vez de la máxima.

5.12 Haga la estimación del retardo de propagación mínimo desde IN hasta OUT para el circuito mostrado en la figura X5.12. Justifique su respuesta.

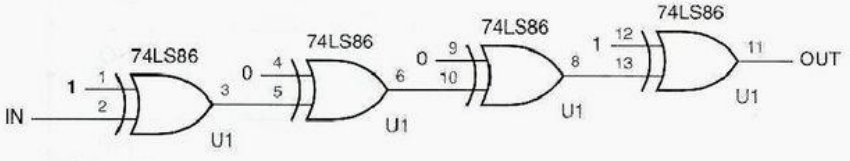


Figura X5.12

5.13 Determine el retardo de propagación máximo exacto desde IN hasta OUT del circuito en la figura X5.12 tanto para transiciones BAJO a ALTO como para ALTO a BAJO, utilizando la información de temporización dada en la tabla 5-2. Repita, con la ayuda de un número de retardo en el peor de los casos simple para cada compuerta, y compare sus resultados. Haga un comentario acerca de estos últimos.

5.14 Repita el problema 5.13, sustituyendo los 74HCT086 por los 74LS86.

5.15 ¿Qué se esperaría que fuera más rápido, un decodificador con salidas alto activas o uno con salidas bajo activas?

5.16 Con la ayuda de la información en la tabla 5-3 para componentes 74LS, determine el retardo de propagación máximo desde cualquier entrada hasta cualquier salida en el circuito decodificador de 5 a 32 de la figura 5-39. Se puede utilizar el método de análisis "del peor de los casos".

5.17 Repita el problema 5.16, efectuando un análisis detallado para cada dirección de transición, y compare sus resultados.

5.18 Dibuje los dígitos creados por un decodificador de siete segmentos 74x49 para las entradas no decimales 1010 hasta 1111.

5.19 Demuestre cómo construir cada una de las siguientes funciones lógicas de salida simple o múltiple, utilizando uno o más decodificadores binarios 74x138 o 74x139 y compuertas NAND. (*Sugerencia:* Cada realización debería ser equivalente a una suma de minitérminos.)

(a) $F = \Sigma_{X,Y,Z}(2,4,7)$

(b) $F = \prod_{A,B,C}(3,4,5,6,7)$

(c) $F = \Sigma_{A,B,C,D}(2,4,6,14)$

(d) $F = \Sigma_{W,X,Y,Z}(0,1,2,3,5,7,11,13)$

(e) $F = \Sigma_{W,X,Y}(1,3,5,6)$
$G = \Sigma_{W,X,Y}(2,3,4,7)$

(f) $F = \Sigma_{A,B,C}(0,4,6)$
$G = \Sigma_{C,D,E}(1,2)$

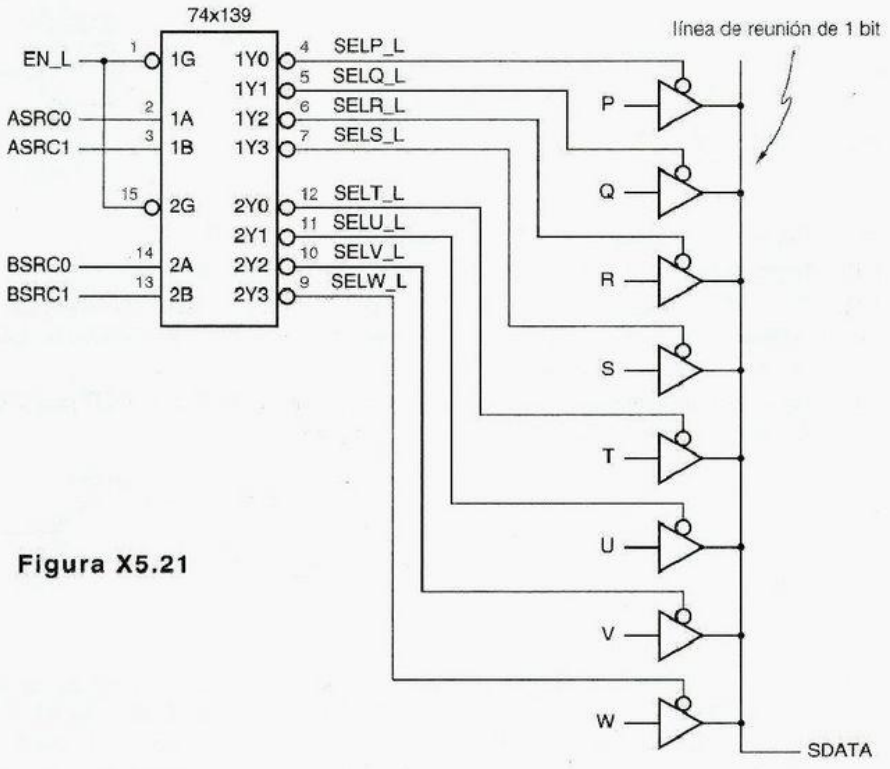


Figura X5.21

5.20 Comenzando con el diagrama lógico para el codificador de prioridad 74x148, escriba las ecuaciones lógicas para sus salidas A2_L, A1_L y A0_L. ¿Cómo difieren de las ecuaciones "genéricas" dadas en la sección 5.5.1?

5.21 ¿Qué terrible equivocación hay en el circuito de la figura X5.21? Sugiera un cambio que elimine ese grave problema.

5.22 Utilizando la información de las tablas 5-2 y 5-3 para los componentes 74LS, determine el retardo de propagación máximo desde cualquier entrada hasta cualquier salida en el circuito multiplexor de 32 a 1 de la figura 5-66. Se puede emplear el método de análisis del "peor de los casos".

5.23 Repita el problema 5.22 utilizando componentes 74HCT.

5.24 Un árbol de paridad de n entradas puede construirse con compuertas XOR en el estilo de la figura 5-74 (a). ¿Bajo qué circunstancias un árbol de paridad con n-entradas (que se forma con compuertas XNOR) realizará exactamente la misma función?

5.25 Con la ayuda de la información en las tablas 5-2 y 5-3 para los componentes 74LS, determine el retardo de propagación máximo desde el bus DU hasta el bus DC en el circuito de corrección de errores de la figura 5-77. Se puede utilizar el método de análisis del "peor de los casos".

5.26 Repita el ejercicio 5.25 utilizando componentes 74HCT.

5.27 Comenzando con las ecuaciones dadas en la sección 5.9.4, escriba una expresión lógica completa para la salida ALTBOUT del 74x85.

5.28 Escriba una expresión algebraica para s_2, el tercer bit de suma de un sumador binario, como una función de entradas x_0, x_1, x_2, y_0, y_1, y y_2. Suponga que $c_0 = 0$ y no intente "distribuir la multiplicación" o minimizar la expresión.

5.29 Comenzando con el diagrama lógico para el 74x682, escriba una expresión lógica para la salida PGTQ_L, en términos de las entradas.

5.30 Con la ayuda de la información en la tabla 5-3 para los componentes 74LS, determine el retardo de propagación máximo desde cualquier entrada hasta cualquier salida del sumador iterativo en grupo de 16 bits de la figura 5-92. Se puede emplear el método de análisis del "peor de los casos".

Ejercicios

5.31 Una posible definición de una compuerta BUT (ejercicio 4.50) es "Y1 es 1 si A1 y B1 son 1 *pero* ya sea A2 o B2 igual a 0; Y2 se define de modo simétrico". Escriba la tabla de verdad y encuentre expresiones de suma de productos mínima para las salidas de la compuerta BUT. Dibuje el diagrama lógico para un circuito NAND-NAND para las expresiones, suponiendo que solamente las entradas no complementadas están disponibles. Se pueden utilizar compuertas de los encapsulados 74x00, '04, '10, '20 y '30.

5.32 Encuentre un diseño al nivel de compuerta para la compuerta BUT definida en el ejercicio 5.31 que utilice un número mínimo de transistores cuando se realiza en CMOS. Se pueden emplear compuertas de los encapsulados 74x00, '02, '04, '10, '20 y '30. Escriba las expresiones de salida (las cuales no necesitan ser producto de sumas de dos niveles) y dibuje el diagrama lógico.

5.33 Para cada circuito en los dos ejercicios anteriores, calcule el retardo en el peor de los casos desde la entrada hasta la salida, utilizando los números de retardo para los componentes 74HCT en la tabla 5-2. Compare el costo (número de transistores), velocidad y carga de entrada de los dos diseños. ¿Cuál será el mejor?

5.34 Implemente la función $F = \Sigma_{W,X,Y,Z}(3,7,11,12,13,14)$ utilizando compuertas BUT. Es decir, muestre cómo efectuar F con una compuerta BUT simple como se definió en el ejercicio 5.31 y una compuerta OR de 2 entradas simple. *implementación de la compuerta but*

5.35 Suponga que un decodificador 74LS138 está conectado de modo que todas las entradas de habilitación se encuentren asertivas y C B a = 101. Empleando la información en la tabla 5-3 y el diagrama lógico interno del '138, determine el retardo de propagación desde la entrada hasta todas las salidas relevantes para cada posible cambio de entrada simple. (*Sugerencia:* existe un total de nueve números de retardo, puesto que un cambio en A, B o C afecta dos salidas, y un cambio en cualquiera de las tres entradas habilitadas afecta una salida.)

5.36 Suponga que se le ha solicitado diseñar un nuevo componente, un decodificador decimal que está optimizado para aplicaciones en las cuales se espera que se presenten solamente combinaciones de entrada decimales. ¿Cómo puede minimizarse el costo de un decodificador de esta clase en comparación con otro que es simplemente un decodificador de 4 a 16 con seis salidas eliminadas? Escriba las ecuaciones lógicas para todas las diez salidas del decodificador minimizado, suponiendo entradas y salidas activas altas y entradas no habilitadas.

5.37 ¿Cuántos mapas de Karnaugh se requerirían para trabajar el ejercicio 5.36 empleando el procedimiento de minimización de salida múltiple formal descrito en la sección 4.3.8?

5.38 Supongamos que un sistema requiere de un decodificador binario de 5 a 32 con una entrada habilitada simple activa baja, semejante a la figura 5-39. Con la entrada EN1 llevada al nivel ALTO, cualquiera de las entradas EN2_L o EN3_L en la figura podrían utilizarse como la habilitada, con la otra entrada aterrizada. Discuta los pros y los contras de emplear EN2_L versus EN3_L.

5.39 Determine si los circuitos de salida a, b y c en el decodificador de siete segmentos corresponden a expresiones de producto de sumas mínimas para estos segmentos, suponiendo que las combinaciones de entrada no decimales son "sin importancia" y BI = 1.

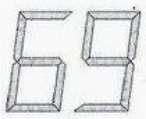


Figura X5.40

5.40 Vuelva a diseñar el decodificador de siete segmentos MSI 74x49 de manera que los dígitos 6 y 9 tengan "cola" como se ilustra en la figura X5.40. ¿Se afecta alguno de los patrones de dígitos para las entradas no decimales 101 hasta la 1111 por su nuevo diseño?

5.41 Comenzando con el programa ABEL en la tabla 5-22, escriba un programa para un decodificador de siete segmentos con las mejoras siguientes:

- Todas las salidas son activas bajas.
- Dos nuevas entradas, ENHEX y ERRDET, controlan la decodificación de salida de segmento.
- Si ENHEX = 0, las salidas coinciden con el comportamiento del 74x49.
- Si ENHEX = 1, entonces las salidas para los dígitos 6 y 9 tienen "cola", y las salidas para los dígitos del A al F están controladas por ERRDET.
- Si ENHEX = 1 y ERRDET = 0, entonces las salidas para los dígitos del A al F tienen el aspecto de las letras A a la F, como en el programa original.
- Si ENHEX = 1 y ERRDET = 1, entonces los dígitos de la A a la F tendrán el aspecto de la letra S.

5.42 Un famoso diseñador lógico decidió abandonar la enseñanza y hacer fortuna otorgando las licencias del diseño de circuito mostrado en la figura X5.42.

(a) Etiquete las entradas y salidas del circuito con nombres de señal apropiados, incluyendo indicaciones del nivel activo.

(b) ¿Qué es lo que hace el circuito? Sea específico y explique todas las entradas y salidas.

(c) Dibuje el símbolo lógico que iría en la hoja de datos de este circuito.

(d) Escriba un programa ABEL o VHDL de comportamiento para el circuito.

(e) ¿Con qué bloques de construcción estándar compite el nuevo circuito? ¿Piensa usted que tendría éxito como una parte MSI?

5.43 Un separador de tres estados FCT controla diez entradas FCT y un resistor de arranque de 4.7 kΩ a 5.0 V. Cuando la salida cambia de BAJO a Hi-Z, haga la estimación de la cantidad de tiempo que transcurre para que las entradas FCT vean la salida de nivel ALTO. Establezca cualquier suposición que haga.

5.44 En un bus de tres estados, 10 separadores FCT de tres estados controlan 10 entradas FCT y un resistor de arranque de 4.7 kΩ a 5.0 V. Suponiendo que ningún otro dispositivo está controlando el bus, haga la estimación de cuánto tiempo permanece la señal del bus en un nivel lógico válido cuando una salida activa entra al estado Hi-Z. Establezca cualquier suposición que haga.

5.45 Diseñe un codificador de 10 a 4 con entradas en el código 1 de 10 y salidas en BCD.

5.46 Dibuje el diagrama lógico para un codificador 16 a 4 utilizando solamente cuatro compuertas NAND de ocho entradas. ¿Cuáles son los niveles activos de las entradas y salidas en su diseño?

5.47 Dibuje el diagrama lógico para un circuito que utiliza el 74x148 para resolver la prioridad entre ocho entradas activas altas, I0 a I7, donde I7 tiene la mayor prioridad. El circuito debería producir salida de dirección activa alta A2 – A0 para indicar el número de la entrada asertiva de mayor prioridad. Si no se afirma ninguna entrada, entonces A2 – A0 deberían ser 111 y debería afirmarse una salida IDLE. Se pueden utilizar compuertas discretas además del '148. Asegúrese de nombrar todas las señales con los niveles activos apropiados.

5.48 Dibuje el diagrama lógico para un circuito que resuelva la prioridad entre ocho entradas activas bajas, I0_L a I7_L, donde I0_L tiene la prioridad más alta. El circuito debería producir salidas de dirección activa altas A2 – A0 para indicar el número de la entrada asertiva de mayor prioridad. Si por lo menos una entrada está asertiva, entonces debería afirmarse una salida AVALID. Asegúrese de nombrar todas las señales con los niveles activos apropiados. Este circuito puede construirse con un solo 74x148 y ninguna otra compuerta más.

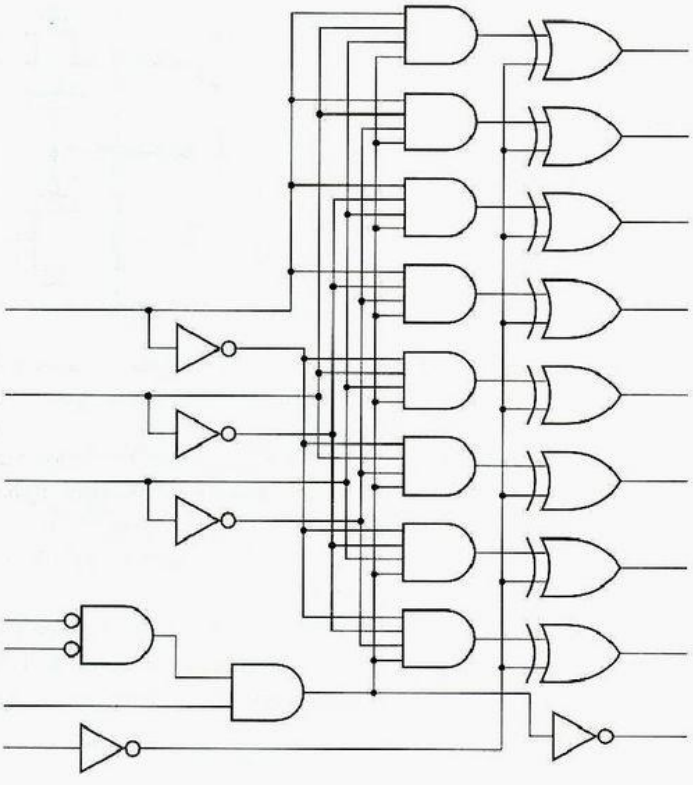


Figura X5.42

5.49 Un propósito del ejercicio 5.48 era demostrar que no siempre es posible mantener la consistencia en la notación de nivel activo a menos que estemos dispuestos a definir símbolos lógicos alternativos para partes MSI que puedan utilizarse en diferentes maneras. Defina un símbolo alterno del 74x148 que proporcione esta consistencia en el ejercicio 5.48.

5.50 Diseñe un circuito combinacional con ocho entradas de solicitud bajo activas, R0_L – R7_L, y ocho salidas, A2 – A0, AVALID, B2 – B0 y BVALID. Las entradas R0_L – R7_L y las salidas A2 – A0 y AVALID se definen como en el ejercicio 5.48. Las salidas B2 – B0 y BVALID identifican la entrada de solicitud con la segunda prioridad más alta que es asertiva. Debería ser capaz de diseñar este circuito con menos de seis paquetes SSI y MSI, pero no utilice más de 10 en cualquier caso.

5.51 Repita el ejercicio 5.50 utilizando ABEL. ¿El diseño queda bien en un solo GAL20V8?

5.52 Repita el ejercicio 5.50 utilizando VHDL.

5.53 Cree un tipo VHDL, basado en IEEE 1164, que modele salidas de colector abierto, donde la escritura de salidas juntas crean una función AND alambrada. También debería modelar un elemento de resistor de arranque tal que si no hay resistor de arranque y ningún dispositivo se encuentra controlando el bus, entonces se produce una señal "desconocida". Pruebe sus definiciones al modelar el circuito en las figuras X3.92 para todas las combinaciones de entrada, tanto con R1 presente como ausente.

5.54 Diseñe un multiplexor de 5 bits y 3 entradas que se ajuste en un encapsulado CI de 24 terminales. Escriba la tabla de verdad y dibuje un diagrama lógico y el símbolo lógico para su multiplexor.

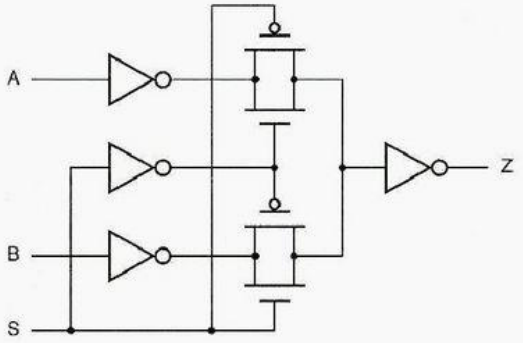


Figura X5.55

5.55 Escriba la tabla de verdad y un diagrama lógico para la función lógica realizada por el circuito CMOS en la figura 5.55. (El circuito contiene compuertas de transmisión, las cuales fueron presentadas en la sección 3.7.1.)

5.56 ¿Qué función lógica es efectuada por el circuito CMOS mostrado en la figura X5.56?

5.57 Un famoso diseñador lógico decidió abandonar la enseñanza y hacer fortuna otorgando las licencias del diseño de circuito mostrado en la figura X5.57.

(a) Etiquete las entradas y salidas del circuito con nombres de señal apropiados, incluyendo indicaciones del nivel activo.

(b) ¿Qué es lo que hace el circuito? Sea específico y explique todas las entradas y salidas.

(c) Dibuje el símbolo lógico que iría en la hoja de datos de este circuito.

(d) Escriba un programa ABEL o VHDL de comportamiento para el circuito.

(e) ¿Con qué bloques de construcción estándar compite el nuevo circuito? ¿Piensa usted que tendría éxito como una parte MSI?

5.58 Escriba un programa VHDL para el multiplexor 74x157 con la tabla de funciones mostrada en la tabla 5-35.

5.59 Escriba un programa VHDL para el multiplexor 74x153 con la tabla de funciones mostrada en la tabla 5-36.

5.60 Demuestre cómo realizar el multiplexor de 18 bits y 4 entradas con la funcionalidad descrita en la tabla 5-40 utilizando 18 dispositivos 74x151.

5.61 Demuestre cómo realizar el multiplexor de 18 bits y 4 entradas con la funcionalidad de la tabla 5-40 usando nueve dispositivos 74x153 y un "convertidor de código" con entradas S2 – S0 y salidas C1, C0 de tal modo que [C1, C0] = 00–11 cuando S2 – S0 seleccionen respectivamente A – D.

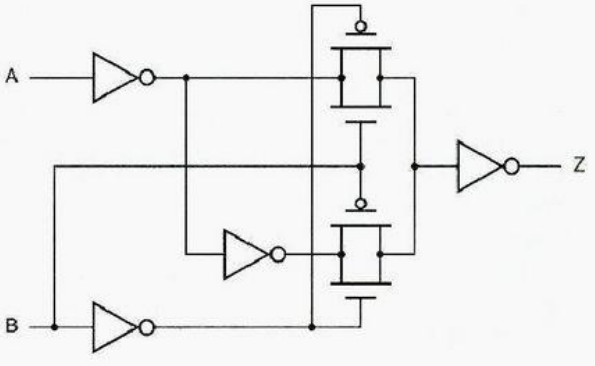


Figura X5.56

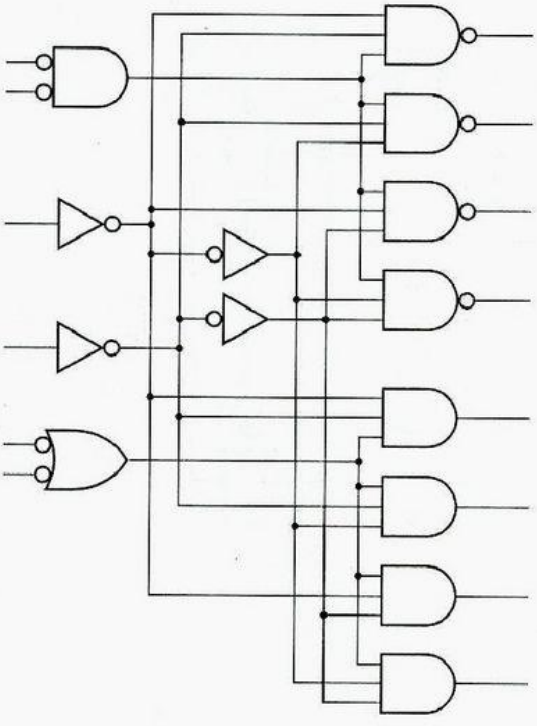


Figura X5.57

5.62 Diseñe un circuito combinacional de 3 entradas y 2 salidas que efectúe la conversión de código especificada en el ejercicio anterior, con la ayuda de compuertas discretas.

5.63 Agregue una entrada de control de salida de tres estados OE al programa multiplexor VHDL en la tabla 5-43. Su solución debería tener solamente un proceso.

5.64 Un *desplazador rápido (traslator en círculo)* de 16 bits en un circuito lógico combinacional con 16 entradas de datos, 16 salidas de datos y 4 entradas de control. La palabra de salida es igual a la palabra de entrada, girada en un número de posiciones de bit especificado por las entradas de control. Por ejemplo, si la palabra de entrada es igual a ABCDEFGHIJKLMNOP (cada letra representa un bit), y las entradas de control son 0101 (5), entonces la palabra de salida FGHIJKLMNOPABCDE. Diseñe un desplazador rápido de 16 bits con la ayuda de partes MSI combinacionales discutidas en este capítulo. Su diseño debería contar con 20 o menos CI. No haga un esquema completo, sino que dibuje y describa su diseño en términos generales e indique los tipos y la cantidad total de CI que se requieren.

desplazador en barril

5.65 Escriba un programa ABEL para el desplazador rápido del ejercicio 5.64.

5.66 Escriba un programa VHDL para el desplazador rápido del ejercicio 5.64.

5.67 Un diseñador digital que construyó el circuito en la figura 5.76 accidentalmente utilizó unos 74x00 en lugar de los '08 en el circuito, y encontró que el circuito todavía funcionaba, excepto por un cambio en el nivel activo de la señal ERROR. ¿Cómo fue esto posible?

5.68 Un circuito de paridad impar con 2^n entradas puede construirse con $2^n - 1$ compuertas XOR. Describa dos estructuras diferentes para este circuito, una de las cuales proporcione un retardo de propagación de entrada a salida mínimo en el peor de los casos y el otro ofrezca un máximo. Para cada estructura, establezca el número del peor de los casos de los retardos de compuerta XOR, y describa una situación donde tal estructura puede ser preferida sobre la otra.

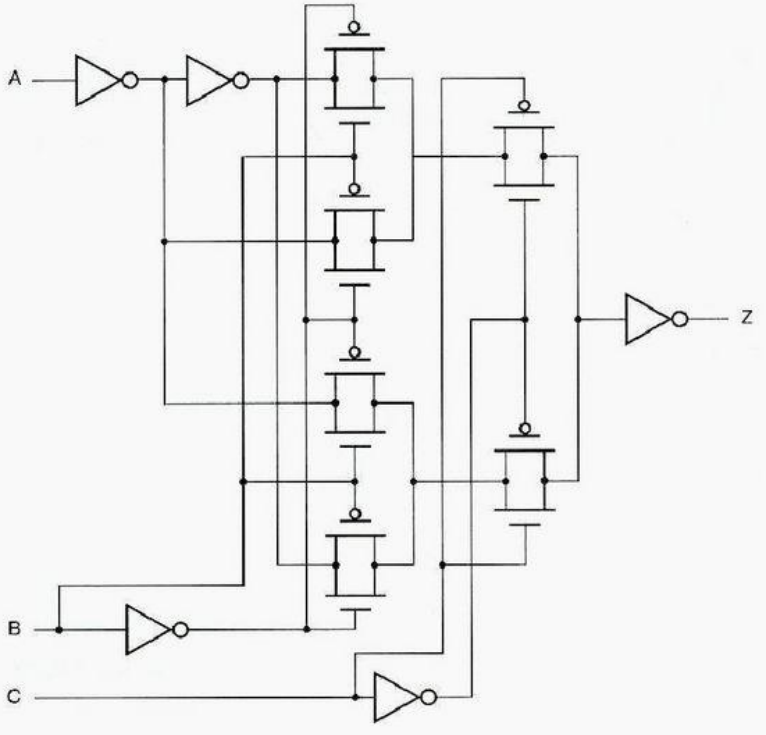


Figura X5.73

5.69 Escriba un algoritmo iterativo de 4 pasos correspondiente al circuito comparador iterativo de la figura 5-80.

5.70 Escriba un programa VHDL para un comparador iterativo de 16 bits empleando la estructura de la figura 5-80. Haga uso de la capacidad "`generate`" de VHDL.

5.71 Diseñe un comparador de 16 bits con la ayuda de cinco 74x85 en una estructura tipo árbol de tal modo que el retardo máximo para una comparación sea igual a dos veces el retardo de un 74x85.

5.72 Escriba un programa VHDL para un dispositivo con la funcionalidad de un 74x85.

5.73 Escriba la tabla de verdad y un diagrama lógico para la función lógica efectuada por el circuito CMOS en la figura X5.73.

5.74 Diseñe un comparador semejante al 74x85 que utilice el orden en cascada opuesto. Es decir, para efectuar una comparación de 12 bits, las salidas en cascada del comparador de orden superior controlaría las entradas en cascada del comparador de orden medio, y las salidas de orden medio controlarían las entradas de orden inferior. No se necesita hacer un diseño y esquema lógicos completos; una tabla de verdad y una nota de aplicación mostrando la interconexión para un comparador de 12 bits serán suficientes.

5.75 Diseñe un comparador de 24 bits utilizando tres 74x682 y compuertas adicionales como sean requeridas. Su circuito debería comparar dos números sin signo P y Q de 24 bits y producir dos bits de salida que indiquen si $P = Q$ o si $P > Q$.

5.76 Con la información de la tabla 5-3, determine el máximo retardo de propagación desde cualquier entrada de bus A o B, hasta cualquier salida de bus F del sumador de acarreo hacia adelante de 16 bits de la figura 5-96. Se puede emplear el método de análisis del "peor de los casos".

5.77 Comenzando con el diagrama lógico para el 74x283 en la figura 5-91, escriba una expresión lógica para la salida S2 en términos de las entradas y demuestre que en realidad iguala el tercer bit de suma en una suma binaria como se afirma. Se puede suponer que $c_0 = 0$ (es decir, ignore c_0).

5.78 Consultando la hoja de datos de un circuito de acarreo anticipado 74LS182, determine si sus salidas satisfacen o no las ecuaciones dadas en la sección 5.10.7.

5.79 Haga la estimación del número de términos de producto en una expresión de producto de sumas mínima para la salida c_{32} de un sumador binario de 32 bits. Sea algo más específico que al decir "miles y miles de millones", y justifique su respuesta.

5.80 Dibuje el diagrama lógico para una ALU de 64 bits utilizando dieciséis 74x181 y cinco 74S182s para acarreo anticipado completo (dos niveles de '182). Para los '181, se necesita mostrar solamente las entradas CIN y las salidas G_L y P_L.

5.81 Escriba un modelo VHDL para una ALU 74x181.

5.82 Muestre cómo construir las cuatro funciones que se presentan a continuación empleando un encapsulado SSI y un 74x138.

$F1 = X' \cdot Y' \cdot Z' + X \cdot Y \cdot Z \qquad F2 = X' \cdot Y' \cdot Z + X \cdot Y \cdot Z'$

$F3 = X' \cdot Y \cdot Z' + X \cdot Y' \cdot Z \qquad F4 = X \cdot Y' \cdot Z' + X' \cdot Y \cdot Z$

5.83 Determine el retardo de propagación en el peor de los casos del multiplicador en la figura 5-98, suponiendo que el retardo de propagación de cualquier entrada de sumador hasta su salida de suma es dos veces el tiempo del retardo a la salida de acarreo. Repita, suponiendo la relación opuesta. Si se estuviera diseñando la celda del sumador a partir de un borrador, ¿qué trayectoria favorecería con el retardo mas breve? ¿Existe un equilibrio óptimo?

5.84 Repita el ejercicio anterior para el multiplicador en la figura 5-99.

5.85 Diseñe un decodificador personalizado con la tabla de función en la tabla X5.85 utilizando partes MSI y SSI. Minimice el número de encapsulados de CI en su diseño.

Tabla X5.85

CS_L	A2	A1	A0	*Salida para aserción*
1	x	x	x	ninguna
0	0	0	x	BILL_L
0	0	x	0	MARY_L
0	0	1	x	JOAN_L
0	0	x	1	PAUL_L
0	1	0	x	ANNA_L
0	1	x	0	FRED_L
0	1	1	x	DAVE_L
0	1	x	1	KATE_L

5.86 Repita el ejercicio 5.85 utilizando ABEL y un solo GAL16V8.

5.87 Repita el ejercicio 5.85 con la ayuda de VHDL.

5.88 Basado en el código de Hamming utilizado en el programa VHDL de la tabla 5-77, escriba un programa VHDL para una entidad de codificador Hamming con entradas de datos de 4 bits y salidas codificadas de datos de 7 bits.

5.89 Empleando ABEL y un GAL16V8 simple, diseñe un multiplexor personalizado con cuatro buses de entrada de 3 bits P, Q, R, T y tres entradas de selección S2 – S0 que elijan uno de los buses para controlar un bus Y de salida de 3 bits, de acuerdo con la tabla X5.89.

Tabla X5.89

S2	S1	S0	*Entrada para selección*
0	0	0	P
0	0	1	P
0	1	0	P
0	1	1	Q
1	0	0	P
1	0	1	P
1	1	0	R
1	1	1	T

5.90 Diseñe un multiplexor personalizado con cuatro buses de entrada de 8 bits P, Q, R y T, seleccionando uno de los buses para controlar un bus Y de salida de ocho bits de acuerdo con la tabla X5.89. Use dos 74x153 y un convertidor de código que mapee los ocho valores posibles en S2 – S0 para cuatro códigos seleccionados para el '153. Elija un código que minimice el tamaño y retardo de propagación del convertidor de código.

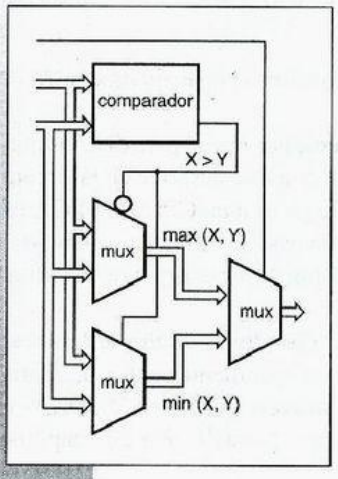
5.91 Diseñe un multiplexor personalizado con cinco buses de entrada de 4 bits A, B, C, D y E, seleccionando uno de los buses para controlar un bus T de salida de 4 bits, de acuerdo con la tabla X5.91. No puede utilizar más de tres CI MSI y SSI.

Tabla X5.91

S2	S1	S0	*Entrada para selección*
0	0	0	A
0	0	1	B
0	1	0	A
0	1	1	C
1	0	0	A
1	0	1	D
1	1	0	A
1	1	1	E

5.92 Repita el ejercicio 5.91 con la ayuda de ABEL y uno o más dispositivos GAL/PAL de este capítulo. Minimice el número y tamaño de los dispositivos GAL.

5.93 Diseñe un verificador de igualdad de 3 bits con seis entradas, SLOT[2-0] y GRANT[2–0] y una salida activa baja. MATCH_L. Las entradas SLOT están conectadas a valores fijos cuando el circuito es instalado en el sistema, pero los valores GRANT cambian de base en un ciclo durante el funcionamiento normal del sistema. Utilizando solamente partes SSI y MSI que aparecen en las tablas 5-2 y 5-3, diseñe un comparador con el retardo de propagación máximo más corto posible desde GRANT[2-0] hasta MATCH_L. (*Nota:* El autor tenía que resolver este problema "en la vida real" para recortar 2 ns el retardo de trayectoria crítica en un diseño de sistema de 25 MHz.)



capítulo 6

Ejemplos de diseño de circuitos combinacionales

Hasta ahora, hemos examinado los principios básicos en varias áreas: sistemas numéricos, circuitos digitales y lógica combinacional, y hemos descrito muchos de los bloques de construcción básicos del diseño combinacional: decodificadores, multiplexores y cosas por el estilo. Todo esto es un fundamento necesario, pero el objetivo último del estudio del diseño digital es finalmente ser capaces de resolver problemas reales mediante el diseño de sistemas digitales (bueno...). Casi siempre, esto requiere más experiencia de la que se puede obtener leyendo un libro. Intentaremos conseguir que usted comience mediante la presentación de un número de ejemplos de diseño combinacional más grandes en este capítulo.

El capítulo está dividido en tres secciones. La primera nos da ejemplos de diseño utilizando bloques de construcción combinacionales. Mientras que los ejemplos son escritos en términos de funciones MSI, las mismas funciones se utilizan en el diseño de dispositivos ASIC y FPGA que están basados en diagramas esquemáticos. La idea de estos ejemplos es mostrar que usted puede expresar a menudo una función combinacional empleando una colección de bloques de construcción más pequeños. Esto es importante por un par de razones: normalmente un enfoque jerárquico simplifica toda la tarea de diseño, y el uso de bloques de construcción más pequeños ofrecen una realización más optimizada, eficaz en FPGA y celdas ASIC que la que se obtendría si se escribiera una descripción monolítica y extensa en un lenguaje de descripción de hardware.

La segunda sección ofrece ejemplos de diseño con ABEL. Estos diseños están dirigidos en su totalidad a pequeños PLD como los 16V8 y 20V8. Aparte del uso general del lenguaje ABEL, algunos de los ejemplos ilustran las decisiones de partición

que debe hacer un diseñador cuando un circuito entero no se ajusta dentro de un solo componente.

Un enfoque basado en VHDL es especialmente apropiado para diseños mayores que serán realizados en un solo CPLD, FPGA o ASIC, como se describe en la tercera sección. Se puede advertir que estos ejemplos no están dirigidos a un CPLD o FPGA específico. En cambio, éste es uno de los beneficios del diseño basado en HDL; la mayoría o la totalidad del esfuerzo de diseño es "transportable" y puede ser dirigido a cualquiera de una variedad de tecnologías.

Los únicos prerrequisitos para este capítulo son los capítulos anteriores. Las tres secciones están escritas de modo que sean muy independientes entre sí, de manera que usted no tenga que leer acerca de ABEL si está interesado sólo en VHDL, o viceversa. También, el resto del libro está escrito de modo que pueda leerse este capítulo ahora o saltarlo y regresar después a él.

6.1 Ejemplos de diseño de bloques de construcción

6.1.1 Desplazador rápido (barrel shifter)

desplazador rápido

Un *desplazador rápido* es un circuito lógico combinacional con n entradas de datos, n salidas de datos y un conjunto de entradas de control que especifican cómo desplazar los datos entre la entrada y la salida. Un desplazador rápido que es parte de un CPU microprocesador típicamente puede especificar la dirección del desplazamiento (izquierda o derecha), el tipo de desplazamiento (circular, aritmético o lógico) y la cantidad de desplazamiento (por lo regular de 0 a $n - 1$ bits, pero en ocasiones de 1 a n bits).

En esta subsección examinaremos el diseño de un desplazador rápido de 16 bits simple que hace solamente desplazamientos circulares a la izquierda, utilizando una entrada de control de 4 bits S[3:0] para especificar la cantidad del desplazamiento. Por ejemplo, si la palabra de entrada es ABCDEFGHGIJHKLMNOP (donde cada letra representa un bit) y la entrada de control es 0101 (5), entonces la palabra de salida es FGHGIHKLMNOPABCDE.

Desde un punto de vista basado en VHDL, este problema es decepcionantemente simple. Cada bit de salida puede obtenerse de un multiplexor de 16 entradas controlado por las entradas de control de desplazamiento. Por el otro lado, cuando examinemos los detalles del diseño usted verá que existen compromisos entre la velocidad y el tamaño del circuito.

Consideremos en primer lugar un diseño que utiliza multiplexores MSI comerciales. Un multiplexor de un bit y 16 entradas puede construirse empleando dos 74x151 aplicando S3 y su complemento a las entradas EN_L y combinando las salidas de datos Y_L con una compuerta NAND, como lo mostramos en la figura 5-66 para un multiplexor de 32 entradas. Las entradas de control de desplazamiento de orden inferior, S2 – S0, se conectan a las entradas de selección de nombre semejante al de los '151.

Completamos el diseño mediante la replicación de este multiplexor de 16 entradas 16 veces y conectando de manera apropiada las entradas de datos, como se muestra en la figura 6-1. El '151 superior de cada par está habilitado mediante S3_L, y el inferior por S3; los bits seleccionados restantes se encuentran conectados a todos los 32 '151. Las entradas de datos D0–D7 de cada '151 están conectadas a las entradas DIN en el orden mencionado de izquierda a derecha.

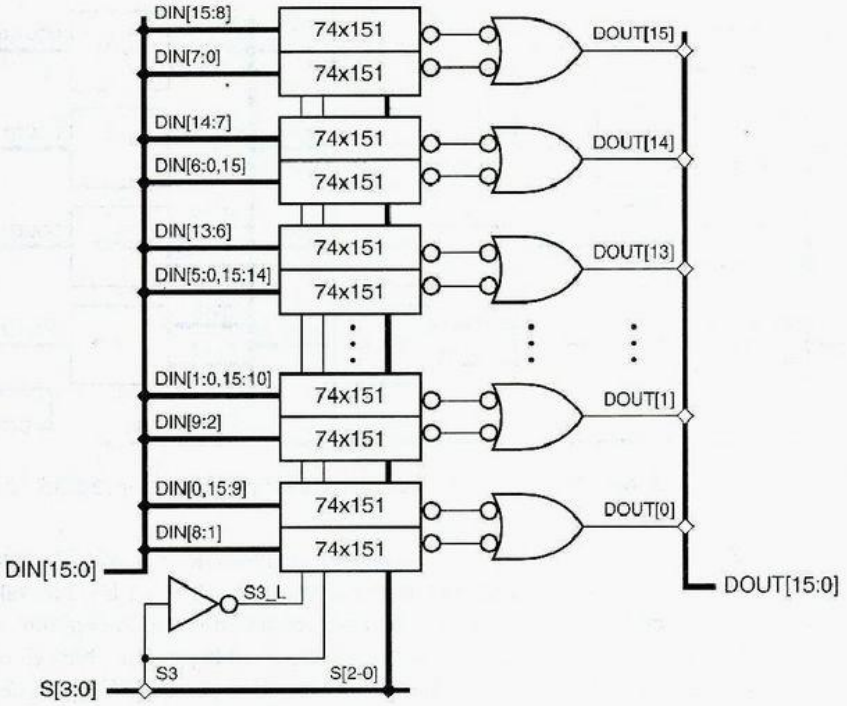


Figura 6-1
Un enfoque para la construcción de un desplazador rápido de 16 bits.

El primer renglón de la tabla 6-1 muestra las características de este primer enfoque. Alrededor de 36 chips (32 74x151, 4 74x00 y 1/6 74x04) se utilizan en la realización MSI/SSI. Podemos reducir esto a 32 chips reemplazando los 74x151 con 74x251 y uniendo sus salidas Y de tercer estado en conjunto, como se tabula en el segundo renglón. Ambos diseños tienen carga muy pesada en las entradas de control; cada uno de los bits de control S[2:0] debe estar conectado a la entrada de selección de nombre semejante en cada uno de los 32 multiplexores. Las entradas de datos también se encuentran con una carga bastante pesada; cada bit de datos debe conectarse a 16 diferentes entradas de datos del multiplexor, correspondientes a las 16 cantidades de desplazamiento posibles. Sin embargo, suponiendo que las pesadas cargas de datos y control no retarden demasiado las cosas, el enfoque basado en el 74x251 produce el retardo de datos más corto, con cada bit de datos pasando a través de sólo un multiplexor.

De manera alternativa, podríamos construir el desplazador rápido utilizando 16 multiplexores 74x157 de 4 bits y 2 entradas, como se tabula en el último renglón de la tabla. Comenzamos utilizando cuatro 74x157 para hacer un multiplexor de 16 bits y

Tabla 6-1
Propiedades de cuatro diferentes enfoques de diseño para desplazadores rápidos.

Componente multiplexor	*Carga de datos*	*Retardo de datos*	*Carga de control*	*CIs totales*
74x151	16	2	32	36
74x251	16	1	32	32
74x153	4	2	8	16
74x157	2	4	4	16

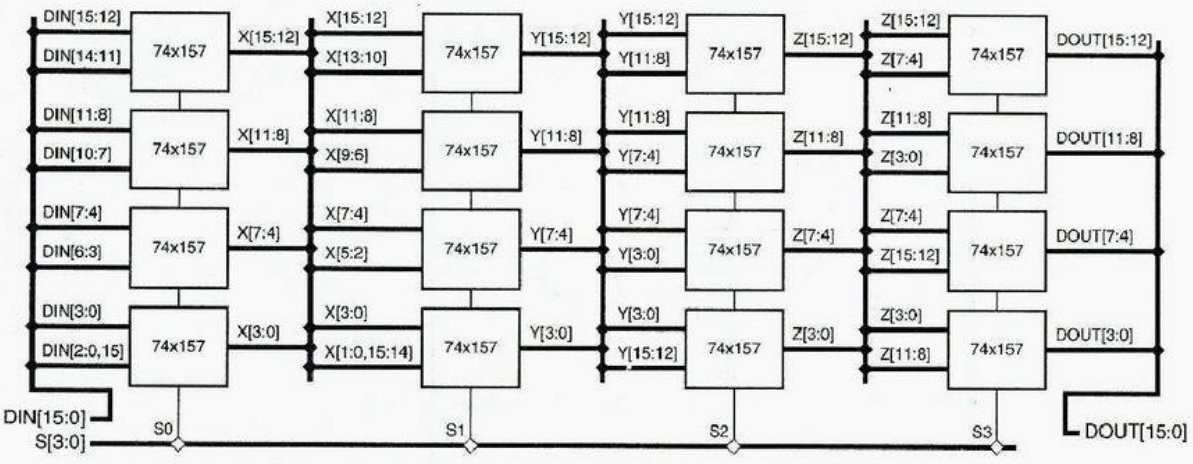


Figura 6-2 Un segundo enfoque para la construcción de un desplazador rápido de 16 bits.

2 entradas. Entonces, podemos conectar un primer conjunto de cuatro '157 controlados por **S0** para desplazar a la izquierda la palabra de entrada por 0 o 1 bit. Las salidas de datos de este conjunto están conectadas a las entradas de un segundo conjunto, controlado por **S1**, que desplaza a la izquierda su palabra de entrada por 0 o 2 bits. Continuando la cascada, un tercer y cuarto conjuntos son controlados por **S2** y **S3** para desplazar selectivamente por 4 y 8 bits, como se muestra en la figura 6-2. Aquí, las entradas **1A-4A** y **1B-4B** y las salidas **1Y-4Y** de cada '157 se encuentran conectadas a las señales indicadas en el orden numerado de izquierda a derecha.

El enfoque basado en el '157 requiere solamente la mitad de paquetes MSI y tiene menor carga sobre las entradas de control y datos. Por otra parte, tiene el retardo de trayectoria de datos más largo, puesto que cada bit de datos debe pasar a través de cuatro 74x157.

A la mitad entre los dos enfoques, podemos utilizar ocho multiplexores 74x153 de 2 bits y 4 entradas para construir un multiplexor de 16 bits y 4 entradas. Poniendo en cascada dos conjuntos de éstos, podemos emplear **S[3:2]** para desplazar de forma selectiva por 0, 4, 8 o 12 bits y **S[1:0]** para desplazar por 0–3 bits. Este enfoque tiene las características de desempeño mostradas en el tercer renglón de la tabla 6-1 y podría aparecer como el mejor compromiso si usted no necesitara tener el retardo de datos absolutamente más corto posible.

La misma clase de consideraciones se aplicaría si usted estuviera construyendo el desplazador rápido con celdas ASIC en vez de partes MSI, excepto que usted estaría contando área de chip en vez de paquetes MSI/SSI.

Las bibliotecas de celda ASIC típicas tienen multiplexores de 1 bit de ancho, por lo regular realizados con compuertas de transmisión CMOS, con 2 a 8 entradas. Para construir un multiplexor más grande se tiene que reunir una combinación apropiada de celdas más pequeñas. Aparte de la clase de selecciones que encontramos en el ejemplo de MSI, se tiene la complicación adicional de que los retardos CMOS son altamente dependientes de la carga. De este modo, dependiendo del enfoque, usted debe decidir dónde agregar buffers a las líneas de control, las líneas de datos o ambas para minimizar los retardos relacionados con la carga. Un enfoque que parece adecuado en teoría, antes de analizar estos retardos y agregar buffers, en la práctica puede generar menos retardo o mayor área de chip que otro.

6.1.2 Codificador de punto flotante simple

En el ejemplo anterior se hizo uso de múltiples copias de un solo bloque de construcción, un multiplexor y era muy obvio del enunciado del problema que un multiplexor sería el bloque de construcción apropiado. El siguiente ejemplo muestra que a veces uno tiene que pensar un poco más para ver la solución en términos de bloques de construcción conocidos.

Examinemos ahora un problema de diseño cuya solución MSI no es tan obvia, un "codificador de punto fijo a punto flotante". Un entero binario sin signo B en el intervalo $0 \leq B < 2^{11}$ puede ser representado por 11 bits en formato de "punto fijo", $B = b_{10}b_9 \ldots b_1b_0$. Podemos representar números en el mismo intervalo con menos precisión usando sólo 7 bits en una notación de punto flotante, $F = M \cdot 2^E$, donde M es una mantisa de 4 bits $m_3m_2m_1m_0$ y E es un exponente de 3 bits $e_2e_1e_0$. El entero más pequeño en este formato es $0 \cdot 2^0$ y el más grande es $(2^4-1) \cdot 2^7$.

Dado un entero B de punto fijo de 11 bits podemos convertirlo en nuestra notación de punto flotante de 7 bits mediante la selección de los cuatro bits de orden mayor comenzando con el 1 más significativo, por ejemplo,

$$\begin{aligned}
11010110100 &= 1101 \cdot 2^7 + 0110100 \\
00100101111 &= 1001 \cdot 2^5 + 01111 \\
00000111110 &= 1111 \cdot 2^2 + 10 \\
00000001011 &= 1011 \cdot 2^0 + 0 \\
00000000010 &= 0010 \cdot 2^0 + 0
\end{aligned}$$

El último término en cada ecuación es un error de truncamiento que resulta de la pérdida de precisión en la conversión. Podemos escribir la especificación para un circuito codificador de punto fijo a punto flotante, correspondiente a esta operación de conversión:

- Un circuito combinacional convierte un entero binario sin signo de 11 bits B en un número de punto flotante de 7 bits M,E, donde M y E tienen 4 y 3 bits, respectivamente. Los números tienen la relación $B = M \cdot 2^E + T$, donde T es el error de truncamiento, $0 \leq T < 2^E$.

Comenzando con un enunciado del problema como el anterior, se requerirá algo de creatividad para llegar a un diseño de circuito eficaz: la sola especificación no da una pista. Sin embargo, podemos obtener algunas ideas examinando cómo convertimos antes números a mano. Básicamente exploramos cada número de entrada de izquierda a derecha para hallar la primera posición conteniendo un 1, deteniéndonos en la posición b_3 si no se encuentra un 1. Seleccionamos cuatro bits iniciando en esa posición para usarlos como la mantisa, y el número de posición de inicio determinaba el exponente. Estas operaciones están comenzando a sonar como bloques de construcción MSI.

Lo que un codificador de prioridad genérico hace es"explorar buscando el primer 1". La salida del codificador de prioridad es un número que nos dice la posición del primer 1. El número de posición determina el exponente; las posiciones del primer 1 de b_{10}–b_3 implican los exponentes de 7–0 y las posiciones de b_2–b_0 o no encontrar el 1 implican un exponente de 0. Por lo tanto, podemos explorar buscando el primer 1 con un codificador de prioridad de 8 entradas con las entradas I7 (la más alta prioridad) hasta I0 conectadas

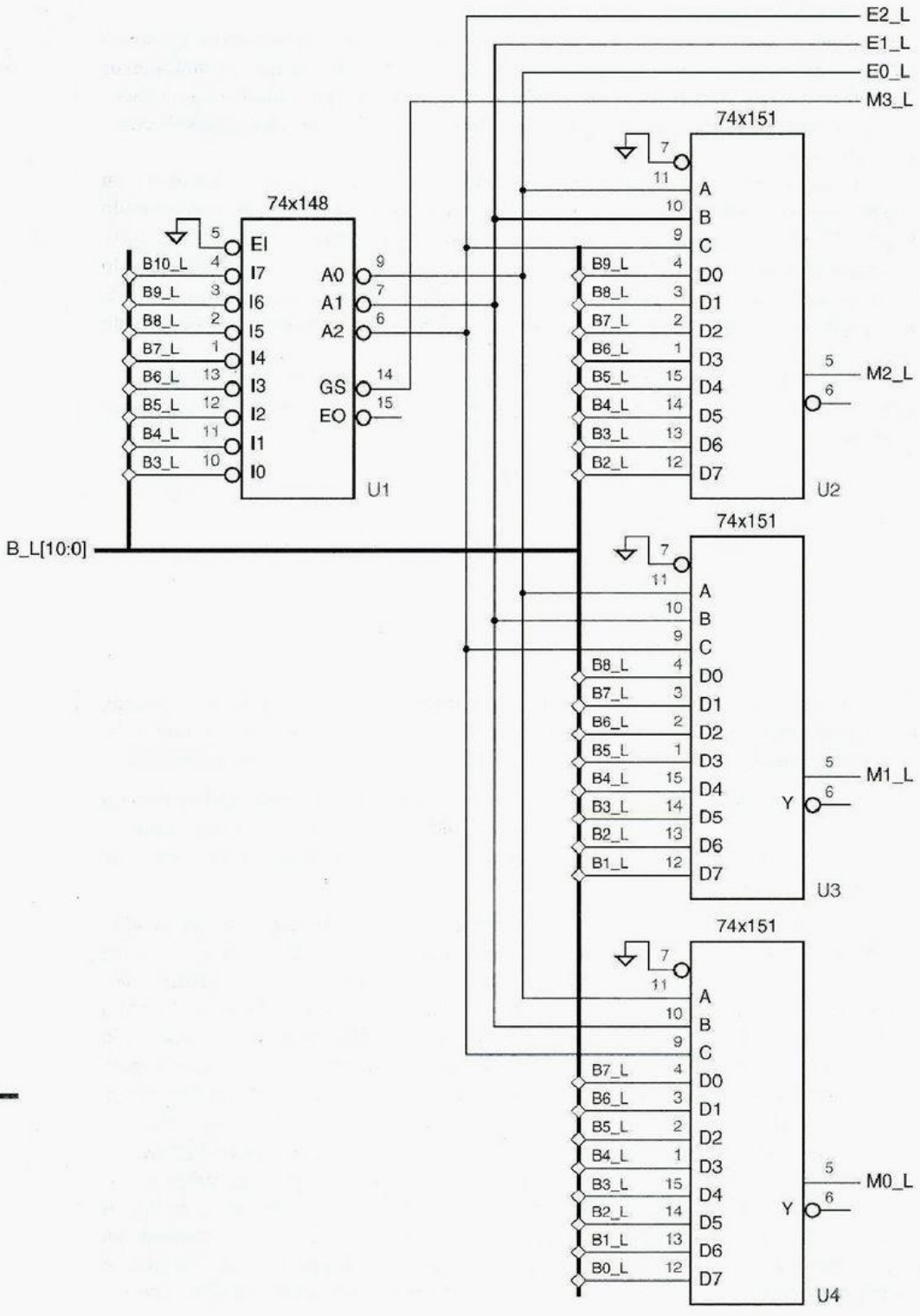


Figura 6-3
Un codificador combinacional de punto fijo a punto flotante.

a b_{10}–b_3. Podemos utilizar las salidas A2–A0 del codificador de prioridad directamente como el exponente mientras que el caso de no hallar el 1 produce A2–A0 = 000.

El hecho de "seleccionar cuatro bits" suena como hacer una "selección" o una operación de multiplexión. El exponente de 3 bits determina cuál conjunto de cuatro bits de B elegiremos, de modo que podemos emplear los bits del exponente para controlar un multiplexor de 4 bits y 8 entradas que selecciona los cuatro bits apropiados de B para formar M.

Un circuito MSI que resulta de estas ideas se ilustra en la figura 6-3. Contiene varias optimizaciones:

- Puesto que el codificador de prioridad MSI disponible, el 74xl48, tiene entradas activas bajas, el número de entrada B se supone que está disponible en un bus activo bajo B_L[10:0]. Si sólo se tiene disponible una versión activa alta de B, entonces pueden utilizarse ocho inversores para obtener la versión activa baja.
- Si reflexiona acerca de la operación de conversión, se dará cuenta de que el bit más significativo de la mantisa, m_3, es siempre 1, excepto en el caso de no hallar el 1. El '148 tiene una salida GS_L que indica este caso, lo que nos permite eliminar el multiplexor para m_3.
- El '148 tiene salidas activas bajas, de modo que los bits del exponente (E0_L–E2_L) son producidos en forma activa baja. Naturalmente, tres inversores podrían utilizarse para producir una versión activa alta.
- Puesto que todo lo demás es activo bajo los bits de mantisa activa baja, se emplean, también. Los bits activos altos también están fácilmente disponibles en las salidas EO_L del '148 y Y_L del '151.

En sentido estricto, los multiplexores en la figura 6-3 están dibujados de manera incorrecta. El símbolo del 74x151 puede dibujarse en forma alternativa como se muestra en la figura 6-4. En palabras, si las entradas de datos del multiplexor son activas bajas, entonces las salidas de datos tienen un nivel activo opuesto al mostrado en el símbolo original. El símbolo de "datos activos bajos" sería preferible en la figura 6-3, puesto que los niveles activos bajos de las entradas y salidas del '151 coincidirían entonces con sus nombres

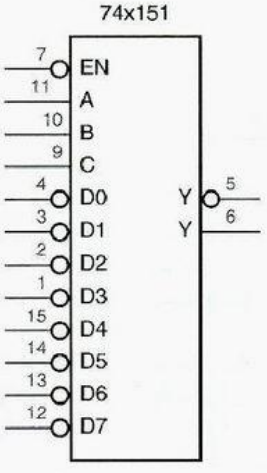


Figura 6-4
Símbolo lógico alterno para el multiplexor de 8 entradas 74x151.

de señal. Sin embargo, en aplicaciones de almacenamiento y transferencia de datos, los diseñadores (y el libro) no siempre "se guían por el manual". En general, por el contexto, un multiplexor (o un registro de bits múltiples, en la sección 8.2.5) no altera el nivel activo de sus datos.

6.1.3 Codificador de prioridad dual

Con bastante frecuencia los bloques de construcción MSI necesitan un poco de ayuda de sus amigos (ordinariamente compuertas) para llevar a cabo el trabajo. En este ejemplo, nos gustaría construir un codificador de prioridad que identifique no sólo la más alta sino también la señal afirmada con la segunda prioridad más alta de entre un conjunto de ocho entradas de solicitud.

Supondremos para este ejemplo que las entradas de solicitud son activas bajas y se llaman R0_L–R7_L donde R0_L tiene la más alta prioridad. Utilizaremos A2-A0 y AVALID para identificar la solicitud de más alta prioridad, donde AVALID es afirmada solamente si por lo menos una entrada de solicitud es afirmada. Emplearemos B2-B0 y BVALID para identificar la solicitud de la segunda prioridad más alta, donde BVALID es afirmada solamente si no por lo menos dos entradas de solicitud son afirmadas.

Encontrar la solicitud de más alta prioridad es bastante fácil; podemos únicamente utilizar un 74xl48. Para hallar la solicitud de segunda más alta prioridad, podemos emplear otro '148, pero sólo si primero "ponemos fuera de combate" la solicitud de más alta prioridad antes de aplicar las entradas de solicitud. Esto puede hacerse utilizando un decodificador para seleccionar una señal para dejar fuera de combate, basado en A2-A0 y AVALID desde el primer '148. Estas ideas son combinadas en la solución mostrada en la figura 6-6. Un decodificador 74x138 afirma como máximo una de sus ocho salidas, correspondientes a la entrada de solicitud de más alta prioridad. Las salidas son alimentadas a una fila de compuertas NAND para "apagar" la solicitud de más alta prioridad.

Un truco que se usa en esta solución es obtener las salidas activas altas desde los '148, como se muestra en la figura 6-5. Podemos renombrar las salidas de dirección A2_L–A0_L para que sean activas altas si también cambiamos el nombre de la entrada de solicitud que está asociada con cada combinación de salida. En particular, complementamos los bits del número de solicitud. En el símbolo redibujado, la entrada de solicitud I0 tiene la prioridad más alta.

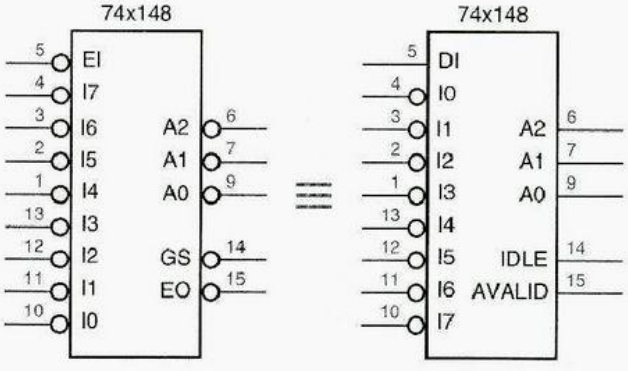


Figura 6-5
Símbolos lógicos alternos para el codificador de prioridad de 8 entradas 74x148.

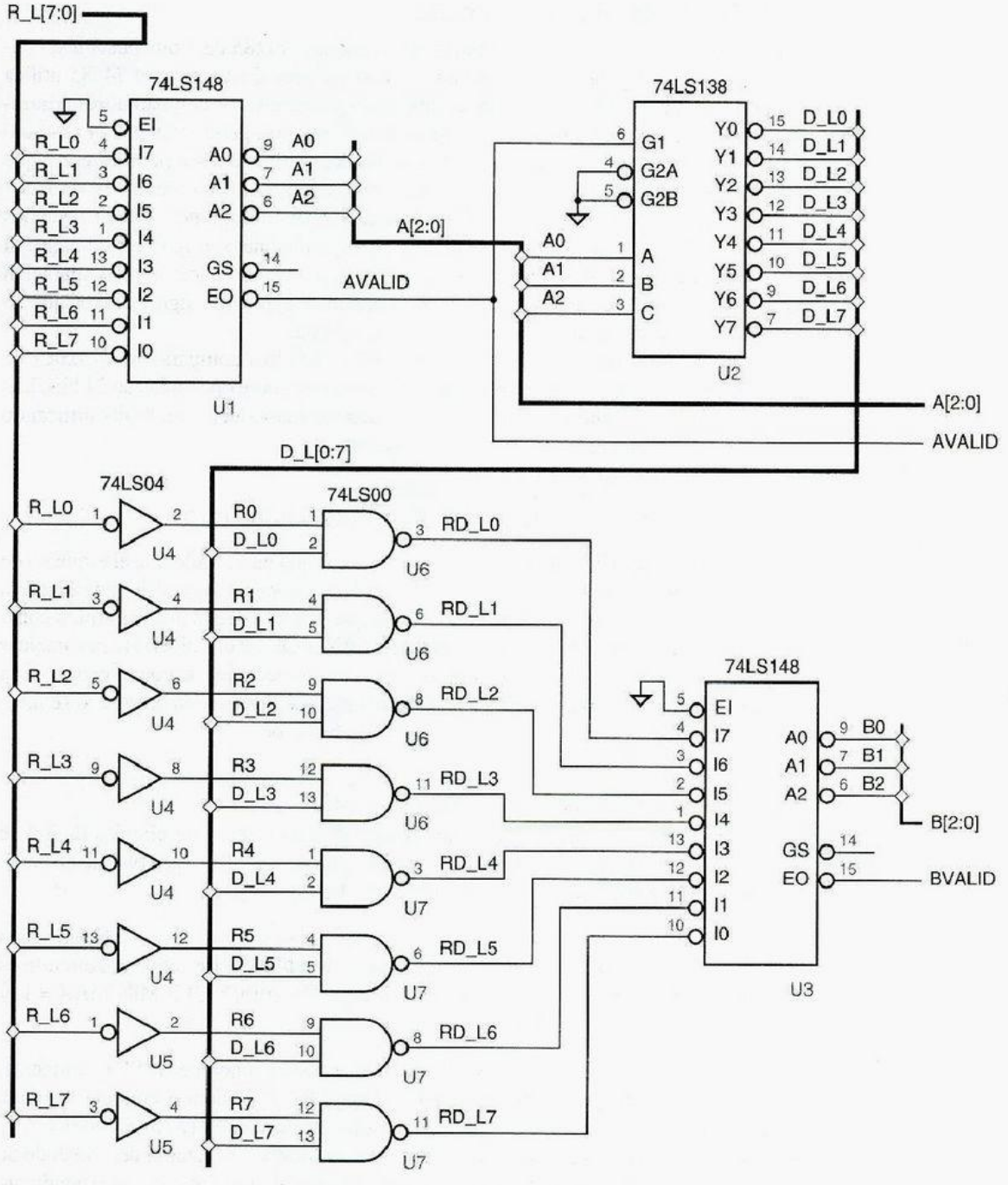


Figura 6-6 Circuito codificador de primera y segunda prioridad más elevada.

6.1.4 Comparadores en cascada

En la sección 5.9.4, mostramos cómo los comparadores 74x85 de 4 bits pueden ser colocados en cascada para crear comparadores más grandes. Puesto que el 74x85 utiliza un esquema en cascada serie, puede ser empleado para construir comparadores arbitrariamente grandes. El comparador 74x682 de 8 bits, por otra parte, no tiene en absoluto ninguna entrada ni salida en cascada. De este modo, a primera vista puede creerse que no es posible utilizarlo para construir comparadores grandes. Pero esto no es verdad.

Si se piensa acerca de la naturaleza de una comparación grande, es claro que dos entradas amplias, digamos de 32 bits (cuatro bytes) cada una, son iguales solamente si sus correspondientes bytes son iguales. Si estamos intentando hacer una comparación mayor que o menor que, entonces los correspondientes bytes más significativos que no sean iguales determinan el resultado de la comparación.

Haciendo uso de estas ideas, la figura 6-7 utiliza tres comparadores 74x682 de 8 bits para hacer comparaciones de igualdad y mayor que sobre operandos de 24 bits. Los resultados de 24 bits son derivados de los resultados individuales de 8 bits utilizando lógica combinacional para las siguientes ecuaciones:

$$PEQQ = EQ2 \cdot EQ1 \cdot EQ0$$
$$PGTQ = GT2 + EQ2 \cdot GT1 + EQ2 \cdot EQ1 \cdot GT0$$

Este enfoque de expansión "paralelo" es en realidad más rápido que el esquema de cascada serie del 74x85, porque no sufre el retardo de propagar las señales en cascada a través de una cascada de comparadores. El enfoque paralelo puede utilizarse para construir comparadores muy anchos empleando lógica AND-OR de dos niveles para combinar los resultados de 8 bits, limitados solamente por las restricciones de convergencia de la lógica AND-OR. Los comparadores arbitrariamente grandes pueden hacerse si se utilizan niveles adicionales de lógica para hacer la combinación.

6.1.5 Comparador dependiente del modo

Con mucha frecuencia, los requerimientos para una aplicación de circuito digital se especifican de una manera que hace obvia la solución de un MSI u otro bloque de construcción. Por ejemplo, considere el siguiente problema:

- Diseñe un circuito combinacional cuyas entradas son dos enteros binarios sin signo de 8 bits, X y Y, y una señal de control MIN/MAX. La salida del circuito es un entero binario sin signo Z de 8 bits tal que Z = min(X,Y) if MIN/MAX = 1, y Z = max(X,Y) si MIN/MAX = 0.

Este circuito es bastante fácil de visualizar en términos de funciones MSI. Claramente, podemos utilizar un comparador para determinar si X > Y. Podemos emplear la salida del comparador para controlar multiplexores que producen min(X,Y) y max(X,Y) y podemos utilizar otro multiplexor para seleccionar uno de estos resultados dependiendo de MIN/MAX. La figura 6-8(a) es el diagrama de bloques de un circuito correspondiente para este enfoque.

Nuestra primera solución se enfoca a las tareas, pero es más costosa de lo necesario. Aunque tiene tres multiplexores de dos entradas, hay nada más dos palabras de entrada, X y Y, que al final se seleccionan y se producen a la salida del circuito. Por lo tanto,

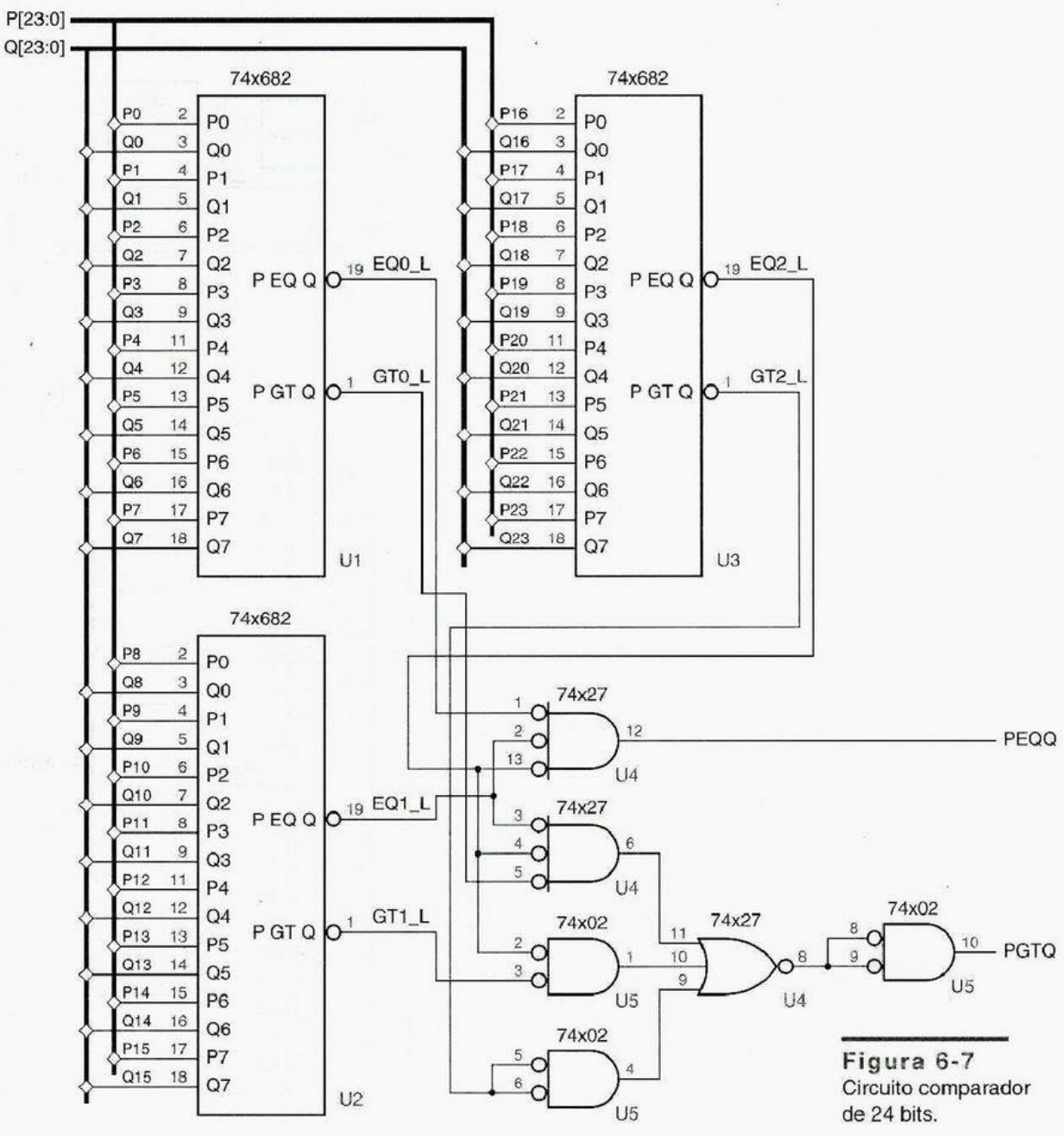


Figura 6-7
Circuito comparador de 24 bits.

deberíamos ser capaces de utilizar justo un multiplexor simple de dos entradas y emplear alguna otra lógica para averiguar qué entrada debe seleccionarse. Este enfoque se ilustra en la figura 6-8(b) y (c). La "otra lógica" es efectivamente muy simple, pues utiliza precisamente una compuerta XOR simple.

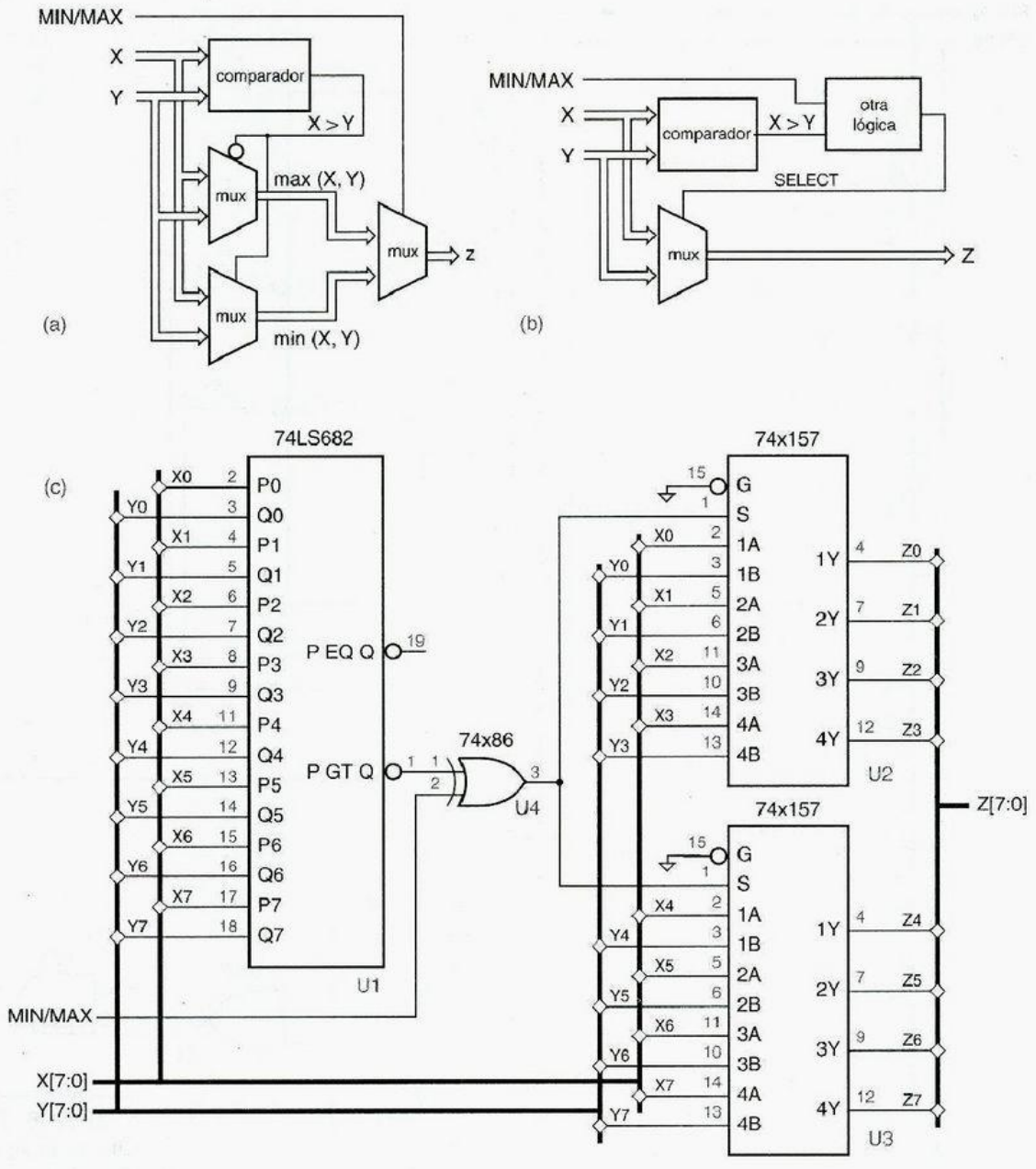


Figura 6-8 Circuito comparador dependiente del modo: a) diagrama de bloques de una solución de "primer-corte"; b) diagrama de bloques de una solución más efectiva en costos; c) diagrama lógico para la segunda solución.

NO SE BLOQUEE

El despilfarro de nuestro enfoque de diseño original en la figura 6-8(a) puede haber sido obvio para usted desde el principio, pero demuestra un enfoque importante para diseñar con bloques de construcción:

- Utilice bloques de construcción estándar para manejar datos, y busque maneras para que un bloque sencillo pueda efectuar diferentes funciones en diferentes tiempos o en diferentes modos. Diseñe circuitos de control para seleccionar las funciones apropiadas como sea necesario, para reducir el conteo de las partes totales del diseño.

Como lo demuestra vehementemente la figura 6-8(c), este enfoque puede ahorrarnos muchos chips. Cuando se diseña con chips de CI, usted *no* debería hacer caso del lema "¡tome todo lo que usted quiera, haremos mucho más!"

6.2 Ejemplos de diseño usando ABEL y PLD

6.2.1 Desplazador rápido

Un desplazador rápido, definido en la página 468, es buen ejemplo de algo que *no* se diseñe utilizando los PDL. Sin embargo, podemos darle buen uso a ABEL para describir una función de desplazador rápido y también veremos por qué un desplazador rápido típico no es una buena adaptación para un PLD.

La tabla 6-2 muestra las ecuaciones para un desplazador rápido de 16 bits con la misma funcionalidad del ejemplo de la página 468 (hace desplazamientos circulares a la izquierda únicamente, utilizando una entrada de control de 4 bits S[3..0] para especificar la cantidad de desplazamiento. ABEL hace fácil especificar la funcionalidad del circuito completo, sin preocuparnos de cómo el circuito puede dividirse en múltiples chips. También, ABEL genera obedientemente una expresión mínima de suma de productos para cada bit de salida. En este caso, cada salida requiere 16 términos de producto.

La división del desplazamiento de 16 bits en múltiples PLD es una tarea difícil en dos maneras diferentes. Primero, debería ser obvio que la naturaleza de la función es tal que todo bit de salida depende de cada bit de entrada. Un PLD que produce, digamos, la salida DOUT0, debe tener todas las 16 entradas DIN y todas las cuatro entradas S disponibles para ello. De este modo, un GAL16V8 en definitiva no puede ser empleado; sólo tiene 16 entradas.

El GAL20V8 es semejante al GAL16V8, con la adición de cuatro terminales sólo de entrada. Si utilizamos la totalidad de las 20 entradas disponibles, nos quedamos con dos terminales sólo de salida (correspondientes a las salidas superior e inferior en la figura 5-27 de la página 345). De este modo, parece posible que podríamos realizar el desplazador rápido utilizando ocho chips 20V8, produciendo dos bits de salida por chip.

No, todavía tenemos un problema. La segunda dimensión de dificultad en un desplazador rápido basado en PLD es el número de términos de producto por salida. El desplazador rápido requiere 16, mientras que el 20V8 proporciona solamente 7. Estamos atorados, ya que cualquier realización del desplazador rápido en los 20V8 va a requerir lógica de paso múltiple. En este punto, estaríamos mejor informados al examinar las opciones de división o partición a lo largo de las líneas, como lo hicimos en la sección 6.1.1.

Tabla 6-2
Programa ABEL para un desplazador rápido de 16 bits.

```
module barrel16
title '16-bit Barrel Shifter'

" Inputs and Outputs
DIN15..DIN0, S3..S0                pin;
DOUT15..DOUT0                      pin istype 'com';

S = [S3..S0];

equations

[DOUT15..DOUT0] = (S==0) & [DIN15..DIN0]
                # (S==1) & [DIN14..DIN0,DIN15]
                # (S==2) & [DIN13..DIN0,DIN15..DIN14]
                # (S==3) & [DIN12..DIN0,DIN15..DIN13]
                ...
                # (S==12) & [DIN3..DIN0,DIN15..DIN4]
                # (S==13) & [DIN2..DIN0,DIN15..DIN3]
                # (S==14) & [DIN1..DIN0,DIN15..DIN2]
                # (S==15) & [DIN0,DIN15..DIN1];

end barrel16
```

El desplazador rápido de 16 bits puede ser realizado sin mucha dificultad en un dispositivo programable más grande, es decir, un CPLD o un FPGA con suficientes terminales de E/S. Sin embargo, imagine que estuviéramos intentando diseñar un desplazador rápido de 32 o 64 bits. Claramente, necesitaríamos utilizar un dispositivo con todavía más terminales de E/S, pero eso no es todo. El número de términos de producto y la gran cantidad de conexiones (todas las entradas conectadas a todas las salidas) aún sería un reto.

En realidad, un montador típico CPLD o FPGA podría tener dificultades realizando un desplazador rápido más grande con retardo pequeño o incluso sin retardo en absoluto. Hay un recurso crítico que tomamos para conceder en los diseños de desplazador rápido de bloque de construcción dividido de la sección 6.1.1: ¡alambres! Un FPGA está algo limitado en su conectividad interna, y un CPLD lo está más aún. De esta forma, incluso con modernas herramientas de diseño FPGA y CPLD, usted todavía tiene que "usar la cabeza" para dividir el diseño de una manera que ayude a las herramientas a efectuar su trabajo.

Los desplazadores rápidos pueden ser todavía más complejos de lo que hemos mostrado hasta ahora. Sólo por diversión, la tabla 6-3 muestra el diseño para un desplazador rápido que respalda seis diferentes clases de desplazamiento. Esto requiere todavía más términos de producto, ¡hasta 40 por salida! Aunque nunca se construiría este dispositivo en un PLD, CPLD o FPGA pequeño, las ecuaciones minimizadas ABEL son útiles porque pueden ayudarle a comprender los efectos de algunas de sus selecciones de diseño. Por ejemplo, al modificar la codificación de `SLA` y `SRA` a `[1,.X.,0]` y `[1,.X.,1]`, se puede reducir el número total de términos de producto en el diseño de 624 hasta 608. Es posible ahorrar más términos de producto al cambiar la codificación de la cantidad de desplazamiento para algunos desplazamientos (véase el ejercicio 6.3). Los ahorros provenientes de estos cambios pueden aprovecharse con otros enfoques de diseño.

Tabla 6-3 Programa ABEL para un desplazador en barril de 16 bits de modo múltiple.

```
module barrl16f
Title 'Multi-mode 16-bit Barrel Shifter'

" Inputs and Outputs
DIN15..DIN0, S3..S0, C2..C0          pin;
DOUT15..DOUT0                        pin istype 'com';

S = [S3..S0]; C = [C2..C0]; " Shift amount and mode
L = DIN15; R = DIN0;        " MSB and LSB

ROL = (C == [0,0,0]); " Rotate (circular shift) left
ROR = (C == [0,0,1]); " Rotate (circular shift) right
SLL = (C == [0,1,0]); " Shift logical left (shift in 0s)
SRL = (C == [0,1,1]); " Shift logical right (shift in 0s)
SLA = (C == [1,0,0]); " Shift left arithmetic (replicate LSB)
SRA = (C == [1,0,1]); " Shift right arithmetic (replicate MSB)

equations

[DOUT15..DOUT0] = ROL & (S==0) & [DIN15..DIN0]
                # ROL & (S==1) & [DIN14..DIN0,DIN15]
                # ROL & (S==2) & [DIN13..DIN0,DIN15..DIN14]
                ...
                # ROL & (S==15) & [DIN0,DIN15..DIN1]
                # ROR & (S==0) & [DIN15..DIN0]
                # ROR & (S==1) & [DIN0,DIN15..DIN1]
                ...
                # ROR & (S==14) & [DIN13..DIN0,DIN15..DIN14]
                # ROR & (S==15) & [DIN14..DIN0,DIN15]
                # SLL & (S==0) & [DIN15..DIN0]
                # SLL & (S==1) & [DIN14..DIN0,0]
                ...
                # SLL & (S==14) & [DIN1..DIN0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                # SLL & (S==15) & [DIN0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
                # SRL & (S==0) & [DIN15..DIN0]
                # SRL & (S==1) & [0,DIN15..DIN1]
                ...
                # SRL & (S==14) & [0,0,0,0,0,0,0,0,0,0,0,0,0,0,DIN15..DIN14]
                # SRL & (S==15) & [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,DIN15]
                # SLA & (S==0) & [DIN15..DIN0]
                # SLA & (S==1) & [DIN14..DIN0,R]
                ...
                # SLA & (S==14) & [DIN1..DIN0,R,R,R,R,R,R,R,R,R,R,R,R,R,R]
                # SLA & (S==15) & [DIN0,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R]
                # SRA & (S==0) & [DIN15..DIN0]
                # SRA & (S==1) & [L,DIN15..DIN1]
                ...
                # SRA & (S==14) & [L,L,L,L,L,L,L,L,L,L,L,L,L,L,DIN15..DIN14]
                # SRA & (S==15) & [L,L,L,L,L,L,L,L,L,L,L,L,L,L,L,DIN15];
end barrl16f
```

6.2.2 Codificador de punto flotante simple

Definimos un formato de número de punto flotante simple en la página 471 y planteamos el problema de diseño de la conversión de un número de punto fijo a su formato de punto flotante. Los requerimientos de la terminal de E/S de este diseño son limitados (11 entradas y 7 salidas) de manera que potencialmente podemos utilizar un PLD sencillo para reemplazar las cuatro partes que hubiéramos utilizado en la solución MSI.

Un programa ABEL para el convertidor de punto fijo a punto flotante se da en la tabla 6-4. La instrucción WHEN expresa la operación de determinar el valor del exponente E de una manera muy natural. Entonces E se emplea para seleccionar los bits apropiados de B para utilizarlo como la mantisa M.

A pesar de la anidación profunda de la instrucción WHEN, solamente cuatro términos de producto se necesitan en la suma mínima para cada bit de E. Las ecuaciones para los M bits no son demasiado malas de todas formas, requiriendo de solamente ocho términos de producto cada una. Por desgracia, el GAL20V8 tiene solamente disponibles siete

Tabla 6-4
Un programa ABEL para el PLD de punto fijo a punto flotante.

```
module fpenc
title 'Fixed-point to Floating-point Encoder'
FPENC device 'P20L8';

" Input and output pins
B10..B0                pin 1..11;
E2..E0, M3..M0         pin 21..15 istype 'com';

" Constant expressions
B = [B10..B0];
E = [E2..E0];
M = [M3..M0];

equations

WHEN B < 16 THEN E = 0;
ELSE WHEN B < 32 THEN E = 1;
ELSE WHEN B < 64 THEN E = 2;
ELSE WHEN B < 128 THEN E = 3;
ELSE WHEN B < 256 THEN E = 4;
ELSE WHEN B < 512 THEN E = 5;
ELSE WHEN B < 1024 THEN E = 6;
ELSE E = 7;

M = (E==0) & [B3..B0]
  # (E==1) & [B4..B1]
  # (E==2) & [B5..B2]
  # (E==3) & [B6..B3]
  # (E==4) & [B7..B4]
  # (E==5) & [B8..B5]
  # (E==6) & [B9..B6]
  # (E==7) & [B10..B7];

end fpenc
```

términos de producto por salida. Sin embargo, el GAL22V10 (figura 8-22 en la página 688) tiene más términos de producto disponibles, de modo que podemos emplear ése si deseamos.

Una desventaja del diseño en la tabla 6-4 es que las salidas `[M3..M0]` son lentas; puesto que utilizan `[E2..E0]`, toman dos pasos a través del PLD. Un enfoque más rápido, si se ajusta, sería reescribir los términos "de selección" (`E==0`, etc.) como ecuaciones intermedias antes de la sección de ecuaciones, y dejar que ABEL expanda las ecuaciones `M` resultantes en un solo nivel de lógica. Por desgracia, ABEL no permite el uso de instrucciones `WHEN` fuera de la sección `equations`, de manera que tendremos que arremangarnos y escribir nuestras propias expresiones lógicas en las ecuaciones intermedias.

La tabla 6-5 muestra el enfoque modificado. Las expresiones para `S7-S0` son justo términos **AND** mutuamente exclusivos que indican valores de exponentes de 7-0 dependiendo de la ubicación del bit 1 más significativo en el número de entrada de punto fijo. El exponente `[E2..E0]` es una codificación binaria de los términos de selección, y los bits de mantisa `[M3..M0]` son generados utilizando un término de selección para cada caso. Resulta que estas `M` ecuaciones todavía requieren de 8 términos de producto por bit de salida, pero al menos son mucho más rápidas, ya que utilizan solamente un nivel de lógica.

Tabla 6-5
Programa ABEL alternativo para el PLD de punto fijo a punto flotante.

```
module fpence
title 'Fixed-point to Floating-point Encoder'
FPENCE device 'P20L8';

" Input and output pins
B10..B0                    pin 1..11;
E2..E0, M3..M0             pin 21..15 istype 'com';

" Intermediate equations
S7 = B10;
S6 = !B10 & B9;
S5 = !B10 & !B9 & B8;
S4 = !B10 & !B9 & !B8 & B7;
S3 = !B10 & !B9 & !B8 & !B7 & B6;
S2 = !B10 & !B9 & !B8 & !B7 & !B6 & B5;
S1 = !B10 & !B9 & !B8 & !B7 & !B6 & !B5 & B4;
S0 = !B10 & !B9 & !B8 & !B7 & !B6 & !B5 & !B4;

equations

E2 = S7 # S6 # S5 # S4;
E1 = S7 # S6 # S3 # S2;
E0 = S7 # S5 # S3 # S1;

[M3..M0] = S0 & [B3..B0] # S1 & [B4..B1] # S2 & [B5..B2]
         # S3 & [B6..B3] # S4 & [B7..B4] # S5 & [B8..B5]
         # S6 & [B9..B6] # S7 & [B10..B7];

end fpenc
```

6.2.3 Codificador de prioridad dual

En este ejemplo, diseñaremos un codificador de prioridad basado en PLD que identifica tanto la señal afirmada de mayor prioridad como la segunda de más alta prioridad de entre un conjunto de ocho entradas de solicitud alto-activas llamadas `[R0..R7]`, donde `R0` tiene la más alta prioridad. Emplearemos `[A2..A0]` y `AVALID` para identificar la solicitud de mayor prioridad, afirmando `AVALID` solamente si se encuentra presente una solicitud de más alta prioridad. De manera similar, utilizaremos `[B2:B0]` y `BVALID` para identificar la solicitud en segundo lugar de mayor prioridad.

La tabla 6-6 muestra un programa ABEL para el codificador de prioridad. Como es habitual, una instrucción anidada `WHEN` es perfecta para expresar el comportamiento de

Tabla 6-6
Programa ABEL para un codificador de prioridad dual.

```
title 'Dual Priority Encoder'
PRIORTWO device 'P16V8';

" Input and output pins
R7..R0                                  pin 1..8;
AVALID, A2..A0, BVALID, B2..B0          pin 19..12 istype 'com';

" Set definitions
A = [A2..A0]; B = [B2..B0];

equations

WHEN R0==1 THEN A=0;
ELSE WHEN R1==1 THEN A=1;
ELSE WHEN R2==1 THEN A=2;
ELSE WHEN R3==1 THEN A=3;
ELSE WHEN R4==1 THEN A=4;
ELSE WHEN R5==1 THEN A=5;
ELSE WHEN R6==1 THEN A=6;
ELSE WHEN R7==1 THEN A=7;

AVALID = ([R7..R0] != 0);

WHEN (R0==1) & (A!=0) THEN B=0;
ELSE WHEN (R1==1) & (A!=1) THEN B=1;
ELSE WHEN (R2==1) & (A!=2) THEN B=2;
ELSE WHEN (R3==1) & (A!=3) THEN B=3;
ELSE WHEN (R4==1) & (A!=4) THEN B=4;
ELSE WHEN (R5==1) & (A!=5) THEN B=5;
ELSE WHEN (R6==1) & (A!=6) THEN B=6;
ELSE WHEN (R7==1) & (A!=7) THEN B=7;

BVALID = (R0==1) & (A!=0) # (R1==1) & (A!=1)
       # (R2==1) & (A!=2) # (R3==1) & (A!=3)
       # (R4==1) & (A!=4) # (R5==1) & (A!=5)
       # (R6==1) & (A!=6) # (R7==1) & (A!=7);

end priortwo
```

```
P-Terms    Fan-in  Fan-out  Type  Name
---------  ------  -------  ----  ----------
   8/1        8        1    Pin   AVALID
   4/5        8        1    Pin   A2
   4/5        8        1    Pin   A1
   4/5        8        1    Pin   A0
  24/8       11        1    Pin   BVALID
  24/17      11        1    Pin   B2
  20/21      11        1    Pin   B1
  18/22      11        1    Pin   B0
=========
 106/84          Best P-Term Total: 76
                       Total Pins: 16
              Average P-Term/Output: 9
```

Tabla 6-7
Uso de término de producto en el PLD codificador de prioridad dual.

prioridad. Para hallar la entrada de prioridad de segunda mayor prioridad, excluimos una entrada, si su número de entrada coincide con el de mayor prioridad, el cual es A: De este modo, estamos empleando lógica de dos pasos para calcular las B salidas. La ecuación para AVALID es fácil; AVALID es 1 si las entradas de solicitud no son todas 0. Para calcular BVALID, operamos con OR todas las condiciones que establecen B en la instrucción WHEN.

Incluso con lógica de dos pasos, las salidas B utilizan tantos términos de producto que no caben en un 16V8; la tabla 6-7 muestra el uso del término de producto. Las salidas B emplean demasiados términos incluso para un 22V10, el cual tiene 16 términos para dos de sus terminales de salida y 8-14 para los otros. ¡En ocasiones usted sólo tiene que trabajar duro para hacer que todo se cumpla!

Así, ¿cómo podemos ahorrar algunos términos de producto? Un punto importante que se tiene que observar es que R0 nunca puede ser la entrada afirmada de segunda más alta prioridad, y por lo tanto B nunca puede ser válida y 0. De este modo, podemos eliminar la cláusula WHEN para el caso R0==1. Haciendo este cambio se reduce el número mínimo de términos para B2-B0 a 14, 17 y 15, respectivamente. Casi podemos ajustar el diseño en un 22V10, si podemos conocer un término fuera de la ecuación B1.

Bien, intentemos algo más. La segunda cláusula WHEN, para el caso R0==2, también falla para hacer uso de todo lo que sabemos. No necesitamos la completa generalidad de

PRODUCTOS DE SUMAS (SE DICE QUE SON 5 VECES MÁS RÁPIDOS)

Usted puede recordar, por la lectura de la sección 4.3.6, que la expresión productos de sumas mínima para el complemento de una función puede ser manipulada a través del teorema de DeMorgan, para obtener una expresión de productos de suma mínima para la función original. También recordará que el número de términos de producto en el producto de sumas mínima puede diferir del número de términos del producto de sumas mínimo. La columna de "P-Terms" en la tabla 6-7 numera el número de términos en ambas formas mínimas (términos de producto/suma). Si *cualquier* forma mínima tiene menos que o igual número de términos de producto disponibles en un arreglo AND-OR de 22V10, entonces la función puede hacerse que ajuste.

A!=0; este caso sólo es importante cuando R0 es 1. Así, reemplacemos las dos primeras líneas de la instrucción WHEN original con

```
WHEN (R1==1) & (R0==1) THEN B=1;
```

Este sutil cambio reduce el número mínimo de términos para B2–B0 a 12, 16 y 13, respectivamente. ¡Lo hicimos! ¿Puede reducirse más el número de términos, suficiente para ajustarse en un 16V8 mientras mantiene la misma funcionalidad? No es probable, pero ¡dejemos eso como un ejercicio (6.4) para el lector!

6.2.4 Comparadores en cascada

En la sección 5.9.5 indicamos que las comparaciones de igualdad son fáciles de realizar en PLD, pero que las comparaciones de magnitud (mayor que o menor que) de más de unos cuantos bits no son buenas candidatas para la realización PLD, obedeciendo al gran número de términos requeridos. De esta forma, los comparadores se realizan mejor utilizando componentes de comparador MSI discretos o como celdas especializadas dentro de una biblioteca FPGA o ASIC. Sin embargo, los PLD son muy adecuados para realizar la lógica combinacional utilizada en esquemas de "expansión en paralelo" que construyen comparadores más amplios de los más pequeños, como mostraremos aquí.

En la sección 5.9.4 presentamos cómo conectar comparadores de 4 bits 74x85 en serie para crear comparadores más grandes. Aunque un esquema en cascada serie no requiere lógica extra para construir de forma arbitraria comparadores más grandes, tiene la desventaja principal de que el retardo se incrementa linealmente con la longitud de la cascada.

En la sección 6.1.4, por otra parte, mostramos cómo múltiples copias del comparador de 8 bits 74x682 podían ser utilizadas en paralelo junto con lógica combinacional para efectuar una comparación de 24 bits. Este esquema puede ser generalizado para comparaciones de amplitud arbitraria.

La tabla 6-8 es un programa ABEL que utiliza un GAL 22V10 para realizar una comparación de 64 bits, utilizando ocho 74x682s para combinar las salidas igual (EQ) y mayor que (GT) del byte individual para producir todas las seis posibles relaciones de los dos valores de entrada de 64 bits ($=, \neq, >, \geq, <, \leq$).

En este programa, cada salida PEQQ y PNEQ se realiza con un término de producto. Las ocho salidas restantes utilizan ocho términos de producto cada una. Como ya mencionamos, el 22V10 suministra 8-16 términos de producto por salida,

HAGA SU PROPIO CAMINO

Los primeros PLD como los PAL16L8 no tenían control de polaridad de salida. Los diseñadores que utilizaban estos dispositivos eran forzados a elegir una polaridad particular, activa alta o activa baja, para algunas salidas a fin de obtener ecuaciones reducidas que pudieran ajustar. Cuando se utiliza un 16V8, 20V8, 22V10 o cualquiera de la amplia variedad de CPLD modernos, no existe tal restricción. Si una ecuación o *su complemento* pueden reducirse al número de términos de producto disponibles, entonces la salida correspondiente puede hacerse activa alta o activa baja, al programar el fusible de polaridad de salida de manera adecuada.

Tabla 6-8
Programa ABEL para combinar ocho dispositivos 74x682 en un comparador de 64 bits.

```
module compexp
title 'Expansion logic for 64-bit comparator'
COMPEXP device 'P22V10';

" Inputs from the individual comparators, active-low, 7 = MSByte
EQ_L7..EQ_L0, GT_L7..GT_L0                pin 1..11, 13..14, 21..23;

" Comparison outputs
PEQQ, PNEQ, PGTQ, PGEQ, PLTQ, PLEQ        pin 15..20 istype 'com';

" Active-level conversions
EQ7 = !EQ_L7; EQ6 = !EQ_L6; EQ5 = !EQ_L5; EQ4 = !EQ_L4;
EQ3 = !EQ_L3; EQ2 = !EQ_L2; EQ1 = !EQ_L1; EQ0 = !EQ_L0;
GT7 = !GT_L7; GT6 = !GT_L6; GT5 = !GT_L5; GT4 = !GT_L4;
GT3 = !GT_L3; GT2 = !GT_L2; GT1 = !GT_L1; GT0 = !GT_L0;

" Less-than terms
LT7 = !(EQ7 # GT7); LT6 = !(EQ6 # GT6); LT5 = !(EQ5 # GT5);
LT4 = !(EQ4 # GT4); LT3 = !(EQ3 # GT3); LT2 = !(EQ2 # GT2);
LT1 = !(EQ1 # GT1); LT0 = !(EQ0 # GT0);

equations

PEQQ = EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & EQ1 & EQ0;

PNEQ = !(EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & EQ1 & EQ0);

PGTQ = GT7  #  EQ7 & GT6  #  EQ7 & EQ6 & GT5
     # EQ7 & EQ6 & EQ5 & GT4  #  EQ7 & EQ6 & EQ5 & EQ4 & GT3
     # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & GT2
     # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & GT1
     # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & EQ1 & GT0;

PLEQ = !(GT7  #  EQ7 & GT6  #  EQ7 & EQ6 & GT5
       # EQ7 & EQ6 & EQ5 & GT4  #  EQ7 & EQ6 & EQ5 & EQ4 & GT3
       # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & GT2
       # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & GT1
       # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & EQ1 & GT0);

PLTQ = LT7  #  EQ7 & LT6  #  EQ7 & EQ6 & LT5
     # EQ7 & EQ6 & EQ5 & LT4  #  EQ7 & EQ6 & EQ5 & EQ4 & LT3
     # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & LT2
     # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & LT1
     # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & EQ1 & LT0;

PGEQ = !(LT7  #  EQ7 & LT6  #  EQ7 & EQ6 & LT5
       # EQ7 & EQ6 & EQ5 & LT4  #  EQ7 & EQ6 & EQ5 & EQ4 & LT3
       # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & LT2
       # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & LT1
       # EQ7 & EQ6 & EQ5 & EQ4 & EQ3 & EQ2 & EQ1 & LT0);

end compexp
```

Tabla 6-9
Bits de control de modo para el comparador dependiente del modo.

M1	*M0*	*Comparación*
0	0	32-bits
0	1	31-bits
1	0	30-bits
1	1	no utilizado

6.2.5 Comparador dependiente del modo

Para el ejemplo siguiente, supondremos que tenemos un sistema en el cual necesitamos comparar dos palabras de 32 bits bajo circunstancias normales, pero donde en ocasiones debemos ignorar uno o dos bits de orden menor de las palabras de entrada. El modo de operación se especifica mediante dos bits de control de modo, M1 y M0, como se muestra en la tabla 6-9.

Como ya lo hemos hecho notar, la comparación, suma y otras operaciones "iterativas" son por lo regular candidatas deficientes para el diseño basado en PLD, porque una expresión equivalente de suma de productos de dos niveles tiene demasiados términos de producto. En la sección 5.9.5 calculamos cuántos términos de producto son necesarios para un comparador de n bits. Con base en estos resultados, con seguridad no estaríamos en la posibilidad de construir el comparador dependiente del modo de 32 bits o incluso una fracción de 8 bits del mismo con un PLD; el comparador de 8 bits 74x682 es apenas aproximadamente el chip sencillo más eficaz posible que podemos utilizar para efectuar una comparación de 8 bits. Sin embargo, un diseño basado en PLD es bastante razonable para manipular la lógica de control de modo y la parte de la comparación que es dependiente del modo (los dos bits de menor orden).

La figura 6-9 muestra un diseño completo de circuito que resulta de esta idea, y la tabla 6-11 es el programa ABEL para un PLD 16V8 MODECOMP que maneja la "lógica aleatoria". Cuatro '682 se utilizan para comparar la mayoría de los bits, y el 16V8 combina las salidas del '682 y manipulan los dos bits de menor orden como una función del modo. Las expresiones intermedias `EQ30` y `GT30` se definen para ahorrar escritura en la sección `equations` del programa.

Como se muestra en la tabla 6-10, las salidas `XEQY` y `XGTY` utilizan 7 y 11 términos de producto, respectivamente. De este modo, `XGTY` no se ajusta dentro de los 7 términos de producto disponibles en una salida 16V8. Sin embargo, éste es otro ejemplo donde tenemos algo de flexibilidad en nuestras selecciones de codificación. Al cambiar la codificación de `MODE30` a `[1,.X.]`, podemos reducir los requerimientos de término de producto para `XGTY` a 7/12 y por ello ajustar el diseño en un 16V8.

Tabla 6-10
Uso de término de producto para el PLD MODECOMP.

```
 P-Terms    Fan-in  Fan-out  Type  Name
-------     ------  -------  ----  --------
   7/9        10        1    Pin   XEQY
  11/13       14        1    Pin   XGTY
=========
  18/22          Best P-Term Total: 18
                       Total Pins: 16
               Average P-Term/Output: 9
```

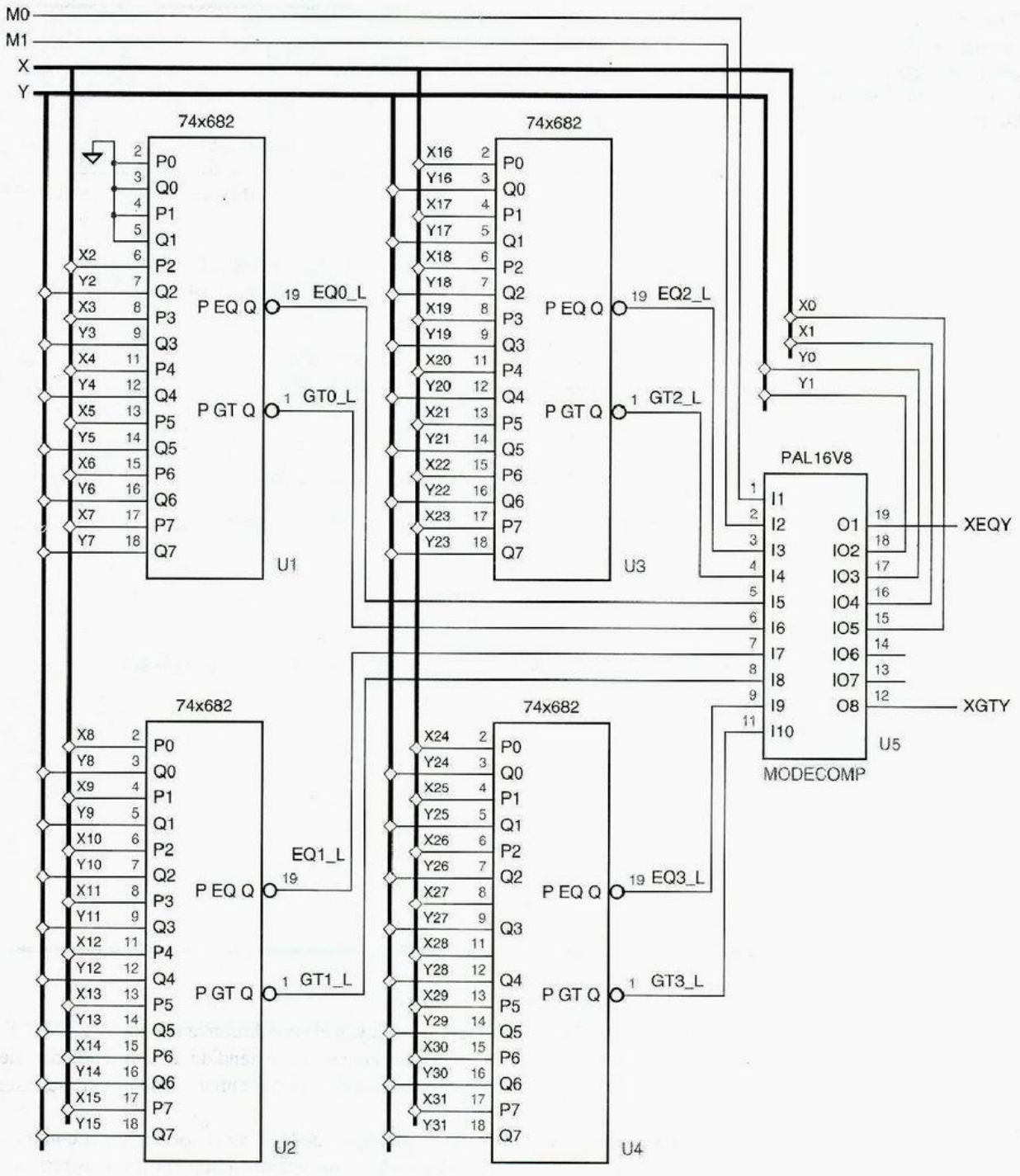


Figura 6-9 Un comparador dependiente del modo de 32 bits.

Tabla 6-11
Programa ABEL para combinar ocho 74x682 en un comparador de 64 bits.

```
module modecomp
title 'Control PLD for Mode-Dependent Comparator'
MODECOMP device 'P16V8';

" Input and output pins
M0, M1, EQ2_L, GT2_L, EQ0_L, GT0_L          pin 1..6;
EQ1_L, GT1_L, EQ3_L, GT3_L, X0, X1, Y0, Y1  pin 7..9, 10, 15..18;
XEQY, XGTY                                  pin 19, 12 istype 'com';

" Active-level conversions
EQ3 = !EQ3_L; EQ2 = !EQ2_L; EQ1 = !EQ1_L; EQ0 = !EQ0_L;
GT3 = !GT3_L; GT2 = !GT2_L; GT1 = !GT1_L; GT0 = !GT0_L;

" Mode definitions
MODE32 = ([M1,M0] == [0,0]); " 32-bit comparison
MODE31 = ([M1,M0] == [0,1]); " 31-bit comparison
MODE30 = ([M1,M0] == [1,0]); " 30-bit comparison
MODEXX = ([M1,M0] == [1,1]); " Unused

" Expressions for 30-bit equal and greater-than
EQ30 = EQ3 & EQ2 & EQ1 & EQ0;
GT30 = GT3 # (EQ3 & GT2) # (EQ3 & EQ2 & GT1) # (EQ3 & EQ2 & EQ1 & GT0);

equations

WHEN MODE32 THEN {
  XEQY = EQ30 & (X1==Y1) & (X0==Y0);
  XGTY = GT30 # (EQ30 & (X1>Y1)) # (EQ30 & (X1==Y1) & (X0>Y0));
  }
ELSE WHEN MODE31 THEN {
  XEQY = EQ30 & (X1==Y1);
  XGTY = GT30 # (EQ30 & (X1>Y1));
  }
ELSE WHEN MODE30 THEN {
  XEQY = EQ30;
  XGTY = GT30;
  }

end modecomp
```

6.2.6 Contador de unos

Existen varios algoritmos importantes que incluyen el paso de contar el número de bits "1" en una palabra de datos. De hecho, recientemente se han extendido algunos conjuntos de instrucciones para microprocesador para incluir el conteo de unos como una instrucción básica.

El conteo de unos en una palabra de datos puede hacerse con facilidad como un proceso iterativo, donde se explora la palabra desde un extremo hasta el otro y se incrementa un contador cada vez que es encontrado un "1". Sin embargo, esta operación debe ser realizada más rápidamente dentro de la unidad lógica y aritmética de un microprocesador. Lo ideal sería que el conteo de unos se ejecutara tan rápido como cualquier otra operación

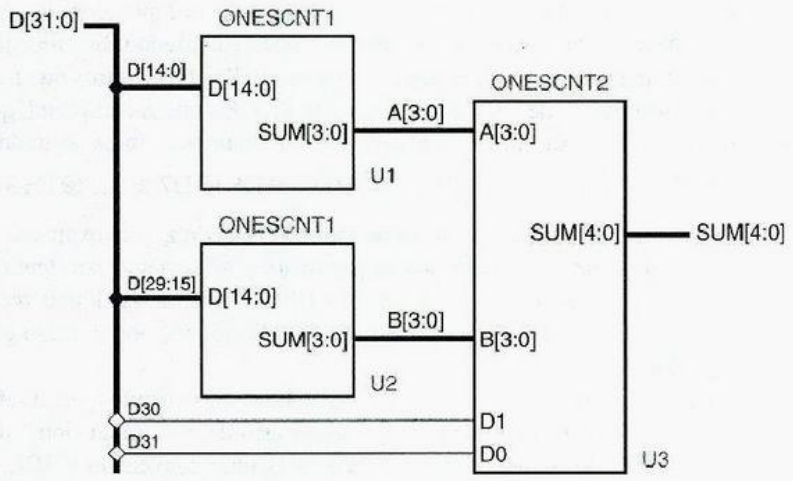


Figura 6-10
Posible partición para el circuito del contador de unos.

aritmética, tal como la suma de dos palabras. Por consiguiente, se requiere de un circuito combinacional.

En este ejemplo, supondremos que tenemos un requerimiento de construir un contador de unos de 32 bits como parte de un sistema mayor. Basados en el número de entradas y salidas requerido, obviamente no podemos ajustar el diseño en un simple PLD clase 22V10, pero podemos ser capaces de dividir el diseño en un número razonablemente pequeño de dispositivos PLD.

La figura 6-10 ilustra una división o partición de este tipo. Dos copias de un primer 22V10, **ONESCNT1**, se utilizan para contar los unos en dos fragmentos de 15 bits de la palabra de 32 bits **D[31:0]**, cada uno con la tarea de producir una salida de suma de 4 bits. Un segundo 22V10, **ONESCNT2**, se utiliza para sumar las sumas de 4 bits y los últimos 2 bits de entrada.

El programa para **ONESCNT1** es simple y decepcionante, como se muestra en la tabla 6-12. La instrucción "`@CARRY 1`" se incluye para limitar la cadena de acarreo a una etapa; como se explicó en la sección 5.10.8, esto reduce los requerimientos del término de producto a expensas de las salidas auxiliares y el retardo incrementado.

Tabla 6-12
Programa ABEL para contar los bits 1 en una palabra de 15 bits.

```
module onescnt1
title 'Count the ones in a 15-bit word'
ONESCNT1 device 'P22V10';

" Input and output pins
D14..D0              pin 1..11, 13..15, 23;
SUM3..SUM0           pin 17..20 istype 'com';

equations

@CARRY 1;
[SUM3..SUM0] = D0 + D1 + D2 + D3 + D4 + D5 + D6 + D7
             + D8 + D9 + D10 + D11 + D12 + D13 + D14;

end onescnt1
```

Por desgracia, cuando yo compilé este programa mi computadora se quedó allí con la CPU ocupada, durante una hora sin producir resultado alguno. Eso me dio tiempo para hacer uso de mi cerebro, un buen ejercicio para aquellos de nosotros que hemos llegado a ser tan dependientes de las herramientas de CAD. Entonces comprendí que podía escribir la función lógica para la salida SUM0 a mano en unos cuantos segundos,

$$SUM0 = D0 \oplus D1 \oplus D2 \oplus D3 \oplus D4 \oplus D5 \oplus D6 \oplus D7 \oplus \ldots \oplus D13 \oplus D14$$

El mapa de Karnaugh para esta función es un tablero de ajedrez, y la expresión mínima del producto de sumas tiene 2^{14} términos de producto. ¡Obviamente esto tendrá cabida en uno o unos cuantos pasos a través de un 22V10! Así que, de cualquier modo, suspendí el proceso de compilación de ABEL y reinicie Windows, por si acaso el compilador hubiera salido mal.

Obviamente, se requiere una partición en fragmentos más pequeños para diseñar el circuito contador de unos. Aunque podríamos perseguir esto de manera adicional utilizando ABEL y PLD, es más interesante hacer un diseño estructural empleando VHDL, como lo haremos en la sección 6.3.6. La versión ABEL y PLD se deja como un ejercicio (6.6).

6.2.7 Juego de "Tres en raya" o "Gato"

En este ejemplo diseñaremos un circuito combinacional que selecciona el siguiente movimiento de un jugador en el juego del "gato" o "tres en raya". La primera cosa que haremos es decidir una estrategia para seleccionar el siguiente movimiento. Intentemos emular la típica estrategia humana siguiendo los pasos de decisión que se plantean a continuación:

1. Busque un renglón, columna o diagonal que tenga dos de mis marcas (X u O, dependiendo de qué jugador sea yo) y una celda vacía. Si existe una, ponga mi marca en la celda vacía; ¡yo gano!
2. Si no es así, busque un renglón, columna o diagonal que tenga dos de las marcas de mi adversario y una celda vacía. Si existe una, ponga mi marca en la celda vacía para bloquear una victoria potencial de mi oponente.
3. Si no, seleccione una celda basándose en la experiencia. Por ejemplo, si la celda de en medio se encuentra abierta, por lo regular es un buen movimiento ocuparla. De otro modo, las celdas de las esquinas son buenas opciones. Los jugadores inteligentes también pueden detectar y bloquear un patrón en desarrollo de parte del adversario o "prever" para seleccionar un buen movimiento.

EL "TRES EN RAYA" O "GATO", EN EL CASO QUE USTED NO LO CONOZCA

El juego de "tres en raya" o "gato" se juega con dos participantes y un tablero con celdas de 3×3 que se encuentran inicialmente vacías. Un jugador es "X" y el otro es "O". Los jugadores se alternan para colocar sus marcas en una celda vacía; "X" siempre empieza primero. El primer jugador en tener tres de sus marcas en el mismo renglón, misma columna o misma diagonal, gana. Aunque el primero que mueve (X) tiene una ligera ventaja, se puede demostrar que un juego entre dos contendientes inteligentes siempre terminará en un empate; ningún jugador tendrá tres en un renglón antes que se llene el tablero.

renglón \ columna	1	2	3
1	X11,Y11	X12,Y12	X13,Y13
2	X21,Y21	X22,Y22	X23,Y23
3	X21,Y21	X23,Y22	X23,Y23

Figura 6-11
Tablero de tres en raya y nombres de señal en ABEL.

Planeando a futuro, podemos llamar al segundo jugador "Y" para evitar la confusión entre "O" y "0" en nuestros programas. La siguiente cosa que hay que pensar es cómo podemos codificar las entradas y salidas del circuito. Existen solamente nueve posibles movimientos que puede efectuar un jugador, de modo que la salida puede ser codificada con sólo cuatro bits. La entrada del circuito es el estado actual del tablero de juego. Tenemos nueve celdas, y cada celda tiene uno de tres estados posibles (vacía, ocupada por una X, ocupada por una Y).

Hay varias opciones de cómo codificar el estado de una celda. Debido a que el juego es simétrico, elegiremos una codificación simétrica que puede ayudarnos posteriormente:

00 La celda está vacía.

10 La celda se encuentra ocupada con una X.

01 La celda se encuentra ocupada con una Y.

De este modo, podemos codificar el estado del tablero de 3×3 en 18 bits. Como se ilustra en la figura 6-11, numeraremos el tablero con números de renglón y columna y utilizaremos las señales `Xij` y `Yij` de ABEL para denotar la presencia de X y Y en la celda `i,j`. Examinaremos la codificación de salida posteriormente.

Con un total de 18 entradas y 4 salidas, el circuito del "tres en raya" podría ajustarse en un dispositivo 22V10. Sin embargo, la experiencia sugiere que no hay modo de hacerlo. Vamos a tener que encontrar una partición de la función, y dividirla siguiendo los pasos de decisión que están en la página anterior, lo cual parece ser una buena idea.

De hecho, los pasos 1 y 2 son muy similares: difieren sólo en que se invierten los papeles del jugador y su adversario. Aquí es donde nuestra codificación simétrica puede

CODIFICACIÓN COMPACTA

Puesto que cada celda en el tablero del tres en raya tiene solamente tres estados, no cuatro, el número total de configuraciones del tablero es 3^9, o 19, 683. Esto es menos que 2^{15}, de modo que el estado del tablero puede ser codificado en solamente 15 bits. Sin embargo, una codificación así conduciría a circuitos mucho más grandes para seleccionar un movimiento, a menos que el circuito de selección de movimientos fuera una memoria sólo de lectura (véase ejercicio 10.26).

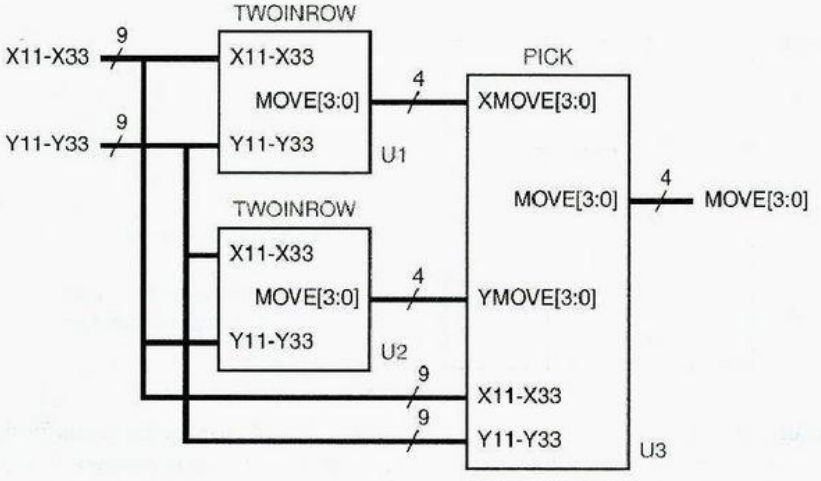


Figura 6-12
División de PLD preliminar para el juego del tres en raya.

valer la pena. Un PLD que me encuentra dos de mis marcas en un renglón junto con una celda vacía para un movimiento ganador (paso 1) puede hallar dos de las marcas de mi oponente en un renglón más una vacía para un movimiento de bloqueo (paso 2). Todo lo que tenemos que hacer es intercambiar las codificaciones para X y Y. Con nuestra codificación seleccionada, eso no requiere ninguna lógica, solamente el intercambio físico entre las señales `Xij` y `Yij` para cada celda. Con esto en mente, podemos utilizar dos copias del mismo PLD, TWOINROW, para efectuar los pasos 1 y 2 como se muestra en la figura 6-12. Nótese que las señales `X11–X33` se encuentran conectadas a las entradas superiores del primer PLD TWOINROW pero a las entradas inferiores del segundo.

Los movimientos de los dos PLD TWOINROW pueden examinarse en otro PLD, PICK. Este dispositivo selecciona un movimiento de los primeros dos PLD si cualquiera encuentra uno; si no, realiza el paso 3. Pareciera que PICK tiene demasiadas entradas y salidas para ajustarse en un 22V10, pero volveremos a eso más tarde.

La tabla 6-13 es un programa para el PLD TWOINROW. Aparece en el estado del tablero desde el punto de vista de X; es decir, busca un movimiento donde X pueda

Tabla 6-13
Programa ABEL para hallar dos en un renglón en el tres en raya.

```
module twoinrow
Title 'Find Two Xs and an empty cell in a row, column, or diagonal'
TWOINROW device 'P22V10';

" Inputs and Outputs
X11, X12, X13, X21, X22, X23, X31, X32, X33 pin 1..9;
Y11, Y12, Y13, Y21, Y22, Y23, Y31, Y32, Y33 pin 10,11,13..15,20..23;
MOVE3..MOVE0                                pin 16..19 istype 'com';

" MOVE output encodings
MOVE   = [MOVE3..MOVE0];
MOVE11 = [1,0,0,0]; MOVE12 = [0,1,0,0]; MOVE13 = [0,0,1,0];
MOVE21 = [0,0,0,1]; MOVE22 = [1,1,0,0]; MOVE23 = [0,1,1,1];
MOVE31 = [1,0,1,1]; MOVE32 = [1,1,0,1]; MOVE33 = [1,1,1,0];
NONE   = [0,0,0,0];
```

Tabla 6-13
(continuación)

```
" Find moves in rows.  Rxy ==> a move exists in cell xy
R11 = X12 & X13 & !X11 & !Y11;
R12 = X11 & X13 & !X12 & !Y12;
R13 = X11 & X12 & !X13 & !Y13;
R21 = X22 & X23 & !X21 & !Y21;
R22 = X21 & X23 & !X22 & !Y22;
R23 = X21 & X22 & !X23 & !Y23;
R31 = X32 & X33 & !X31 & !Y31;
R32 = X31 & X33 & !X32 & !Y32;
R33 = X31 & X32 & !X33 & !Y33;

" Find moves in columns.  Cxy ==> a move exists in cell xy
C11 = X21 & X31 & !X11 & !Y11;
C12 = X22 & X32 & !X12 & !Y12;
C13 = X23 & X33 & !X13 & !Y13;
C21 = X11 & X31 & !X21 & !Y21;
C22 = X12 & X32 & !X22 & !Y22;
C23 = X13 & X33 & !X23 & !Y23;
C31 = X11 & X21 & !X31 & !Y31;
C32 = X12 & X22 & !X32 & !Y32;
C33 = X13 & X23 & !X33 & !Y33;

" Find moves in diagonals.  Dxy or Exy ==> a move exists in cell xy
D11 = X22 & X33 & !X11 & !Y11;
D22 = X11 & X33 & !X22 & !Y22;
D33 = X11 & X22 & !X33 & !Y33;
E13 = X22 & X31 & !X13 & !Y13;
E22 = X13 & X31 & !X22 & !Y22;
E31 = X13 & X22 & !X31 & !Y31;

" Combine moves for each cell.  Gxy ==> a move exists in cell xy
G11 = R11 # C11 # D11;
G12 = R12 # C12;
G13 = R13 # C13 # E13;
G21 = R21 # C21;
G22 = R22 # C22 # D22 # E22;
G23 = R23 # C23;
G31 = R31 # C31 # E31;
G32 = R32 # C32;
G33 = R33 # C33 # D33;

equations

WHEN G22 THEN MOVE= MOVE22;
ELSE WHEN G11 THEN MOVE = MOVE11;
ELSE WHEN G13 THEN MOVE = MOVE13;
ELSE WHEN G31 THEN MOVE = MOVE31;
ELSE WHEN G33 THEN MOVE = MOVE33;
ELSE WHEN G12 THEN MOVE = MOVE12;
ELSE WHEN G21 THEN MOVE = MOVE21;
ELSE WHEN G23 THEN MOVE = MOVE23;
ELSE WHEN G32 THEN MOVE = MOVE32;
ELSE MOVE = NONE;

end twoinrow
```

obtener tres en raya. El programa hace uso extenso de ecuaciones intermedias para definir todos los posibles movimientos de renglón, columna y diagonal. Combina todos los movimientos para una celda `i, j` en una expresión para `Gij`, y finalmente la sección `equations` utiliza una instrucción `WHEN` para seleccionar un movimiento.

Advierta que debe utilizarse una instrucción anidada `WHEN` más que nueve asignaciones o instrucciones `WHEN` en paralelo, porque podemos seleccionar solamente un movimiento incluso si se tienen disponibles múltiples movimientos. También observe que `G22`, la celda central, se verifica en primer lugar, seguida de las esquinas. Esto se hace esperando que podríamos minimizar el número de términos al poner, primero, los movimientos más comunes en el `WHEN` anidado. Por desgracia, el diseño requiere todavía un gran número de términos de producto, como se muestra en la tabla 6-14.

Por cierto, aún no hemos explicado por qué elegimos la codificación de salida que hicimos (como se define mediante `MOVE11`, `MOVE22`, etc., en el programa). Es bastante claro que el cambio de la codificación nunca nos llevaría a ahorrarnos suficientes términos de producto para ajustar el diseño en un 22V10. Pero todavía hay métodos para llegar a esta locura, como lo demostraremos ahora.

Evidentemente, tendremos que dividir TWOINROW en dos o más segmentos. Como en muchos problemas de diseño, se pueden aplicar estrategias diferentes. La primera estrategia que intenté fue utilizar dos PLD diferentes, uno para hallar movimientos en todos los renglones y uno de las diagonales, y el otro para trabajar en todas las columnas y la diagonal restante. Eso ayuda, pero no lo suficiente para ajustar cada mitad en un 22V10.

Con la segunda estrategia, intenté segmentar el problema de una manera distinta. El primer PLD encuentra todos los movimientos en las celdas 11, 12, 13, 21 y 22, mientras que el segundo PLD los encuentra en las celdas restantes. ¡Eso funcionó! El primer PLD, llamado TWOINHAF, se obtiene de la tabla 6-13 simplemente al dejar como comentarios las cuatro líneas de la instrucción `WHEN` para los movimientos de las celdas 23, 31, 32 y 33.

Podríamos obtener el segundo PLD de TWOINROW de una manera similar, pero esperen un minuto. En la fabricación de sistemas digitales reales, siempre es deseable minimizar el número de partes distintas que serán utilizadas; esto ahorra costos de inventario y complejidad. Con partes programables, es deseable que se minimice el número de programas distintos que son utilizados. Aun cuando las partes componentes sean (físicamente) idénticas, se debe aplicar un conjunto diferente de vectores de prueba (a algún costo) para cada programa diferente. También, es posible que el producto vaya a tener éxito suficiente para nosotros para ahorrar dinero al convertir los PLD en dispositivos codificados en circuitos, uno diferente para cada programa, alentándonos de nuevo a minimizar programas.

Tabla 6-14
Uso de término de producto para el PLD TWOINROW.

```
 P-Terms    Fan-in  Fan-out  Type  Name
---------   ------  -------  ----  -----------
  61/142      18        1    Pin   MOVE3
 107/129      18        1    Pin   MOVE2
  77/88       17        1    Pin   MOVE1
 133/87       18        1    Pin   MOVE0
=========
 378/446          Best P-Term Total: 332
                        Total Pins: 22
                Average P-Term/Output: 83
```

El juego del "tres en raya" es el mismo juego incluso si giramos el tablero 90° o 180°. Así, el PLD TWOINHAF puede hallar movimientos para las celdas 33, 32, 31, 23 y 22 si giramos el tablero 180°. Debido a la manera en que definimos el estado del tablero, con un par separado de entradas para cada celda, podemos "girar el tablero" sencillamente al mezclar los pares de entradas de manera apropiada. Es decir, 33↔11, 32↔12, 31↔13 y 23↔21.

Por supuesto, una vez que reacomodamos las entradas, TWOINHAF aún producirá códigos de movimientos de salida que corresponden a las celdas de la parte superior del tablero. Para que todo salga bien, deberíamos transformar éstas en códigos para las celdas apropiadas en la mitad inferior del tablero. Nos gustaría que esta transformación tomara un mínimo de lógica. Aquí es donde nuestra selección del código de salida se hace presente. Si se observó cuidadosamente el código MOVE definido al principio de la tabla 6-13, se verá que el código para una posición dada en el tablero girado 180° se obtiene al complementar e invertir el orden de los bits de código para la misma posición en el tablero sin girar. En otras palabras, la transformación de código puede hacerse con cuatro inversores y un reacomodo de los alambres. Esto puede hacerse "gratis" en el PLD que examina las salidas de TWOINHAF.

Probablemente usted nunca pensó que el "tres en raya" podría ser tan difícil. Bien, nos encontramos a la mitad del camino. La figura 6-13 ilustra la partición del diseño como ahora lo continuaremos. Cada PLD TWOINROW de nuestra partición original es reemplazado por un par de PLD TWOINHAF. EL PLD inferior está precedido por un

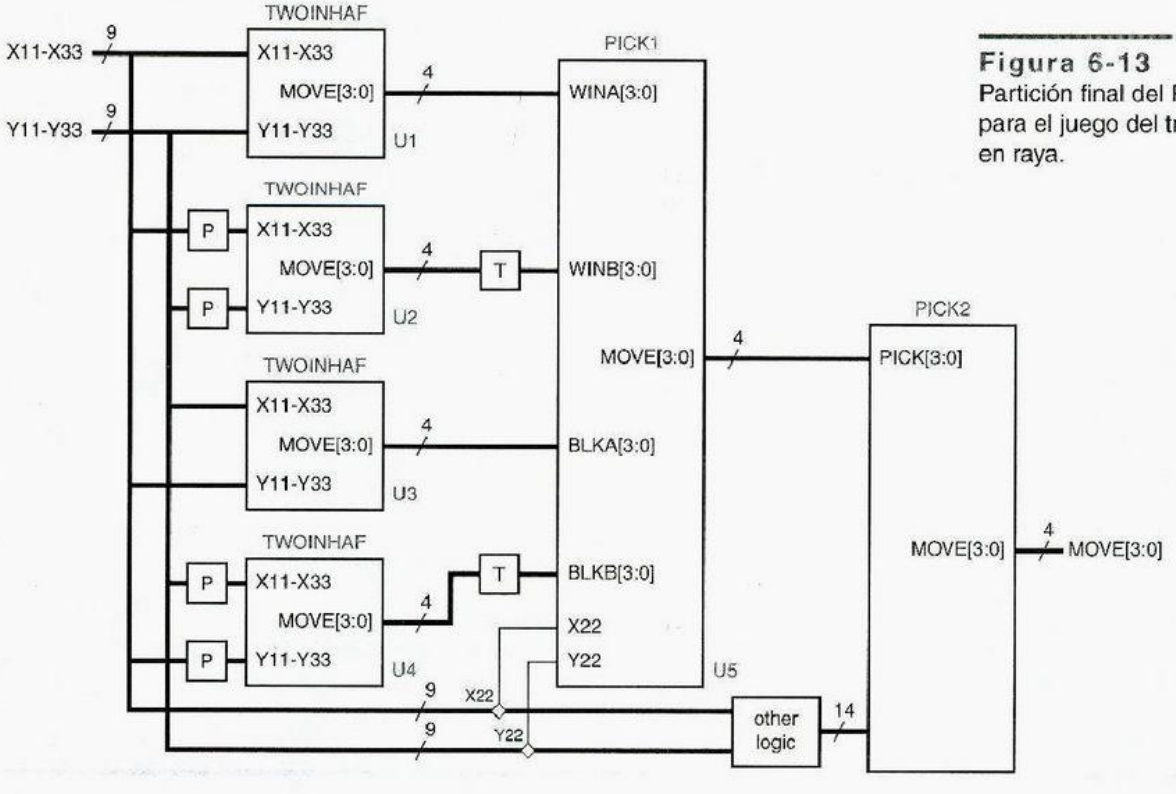


Figura 6-13
Partición final del PLD para el juego del tres en raya.

cuadro etiquetado "P", el cual permuta las entradas para girar el tablero 180°, como lo discutimos con anterioridad. Igualmente, es seguido por un cuadro etiquetado "T", que compensa la rotación mediante la transformación del código de salida; este cuadro en realidad será absorbido en el PLD que sigue, PICK1.

La función de PICK1 es muy directa. Como se mostró en la tabla 6-15, simplemente selecciona un movimiento ganador o un movimiento de bloqueo si se tiene uno disponible. Puesto que existen dos terminales de entrada extra disponibles en el 22V10, los utilizamos para introducir el estado de la celda central. De esta manera, podemos realizar la primera parte del paso 3 del algoritmo "humano" de la página 492, para seleccionar la celda central cuando no hay disponible ningún movimiento ganador o de bloqueo. El PICK1 PLD utiliza al menos 9 términos de producto por salida.

Tabla 6-15 Programa ABEL para seleccionar un movimiento basado en cuatro entradas.

```
module pick1
Title 'Pick One Move from Four Possible'
PICK1 device 'P22V10';

" Inputs from TWOINHAF PLDs
WINA3..WINA0      pin 1..4;       "Winning moves in cells 11,12,13,21,22
WINB3..WINB0      pin 5..8;       "Winning moves in cells 11,12,13,21,22 of rotated grid
BLKA3..BLKA0      pin 9..11, 13;  "Blocking moves in cells 11,12,13,21,22
BLKB3..BLKB0      pin 14..16, 21; "Blocking moves in cells 11,12,13,21,22 of rotated grid
" Inputs from grid
X22, Y22          pin 22..23;     "Center cell; pick if no other moves
" Move outputs to PICK2 PLD
MOVE3..MOVE0      pin 17..20 istype 'com';

" Sets
WINA = [WINA3..WINA0];  WINB = [WINB3..WINB0];
BLKA = [BLKA3..BLKA0];  BLKB = [BLKB3..BLKB0];
MOVE = [MOVE3..MOVE0];

" Non-rotated move input and output encoding
MOVE11 = [1,0,0,0]; MOVE12 = [0,1,0,0]; MOVE13 = [0,0,1,0];
MOVE21 = [0,0,0,1]; MOVE22 = [1,1,0,0]; MOVE23 = [0,1,1,1];
MOVE31 = [1,0,1,1]; MOVE32 = [1,1,0,1]; MOVE33 = [1,1,1,0];
NONE   = [0,0,0,0];

equations

WHEN WINA != NONE THEN MOVE = WINA;
ELSE WHEN WINB != NONE THEN MOVE = ![WINB0..WINB3];  " Map rotated coding
ELSE WHEN BLKA != NONE THEN MOVE = BLKA;
ELSE WHEN BLKB != NONE THEN MOVE = ![BLKB0..BLKB3];  " Map rotated coding
ELSE WHEN !X22 & !Y22 THEN MOVE = MOVE22;            " Pick center cell if it's empty
ELSE MOVE = NONE;

end pick1
```

La parte final del diseño de la figura 6-13 es el PLD PICK2. Este PLD debe proporcionar la mayoría de la "experiencia" en el paso 3 del algoritmo humano si PICK1 no encuentra un movimiento.

Tenemos un pequeño problema con PICK2 en el sentido que un 22V0 no tiene suficientes terminales para acomodar la entrada de 4 bits de PICK1, su propia salida de 4 bits y la totalidad de los 18 bits del estado del tablero; tiene solamente 22 terminales de E/S. En realidad, no necesitamos conectar X22 y Y22, porque ya fueron examinados en PICK1, pero eso todavía nos deja con dos terminales menos. De este modo, el propósito del bloque de "otra lógica" en la figura 6-13 es codificar algo de la información para

Tabla 6-16 Programa ABEL para seleccionar un movimiento empleando la "experiencia".

```
module pick2
Title 'Pick a move using experience'
PICK2 device 'P22V10';

" Inputs from PICK1 PLD
PICK3..PICK0                              pin 1..4;  " Move, if any, from PICK1 PLD
" Inputs from Tic-Tac-Toe grid corners
X11, Y11, X13, Y13, X31, Y31, X33, Y33    pin 5..11, 13;
" Combined inputs from external NOR gates; 1 ==> corresponding cell is empty
E12, E21, E23, E32                        pin 14..15, 22..23;
" Move output
MOVE3..MOVE0                              pin 17..20 istype 'com';

PICK = [PICK3..PICK0];  " Set definition
" Non-rotated move input and output encoding
MOVE = [MOVE3..MOVE0];
MOVE11 = [1,0,0,0]; MOVE12 = [0,1,0,0]; MOVE13 = [0,0,1,0];
MOVE21 = [0,0,0,1]; MOVE22 = [1,1,0,0]; MOVE23 = [0,1,1,1];
MOVE31 = [1,0,1,1]; MOVE32 = [1,1,0,1]; MOVE33 = [1,1,1,0];
NONE   = [0,0,0,0];

" Intermediate equations for empty corner cells
E11 = !X11 & !Y11;  E13 = !X13 & !Y13;  E31 = !X31 & !Y31;  E33 = !X33 & !Y33;

equations

"Simplest approach -- pick corner if available, else side
WHEN PICK != NONE THEN MOVE = PICK;
ELSE WHEN E11 THEN MOVE = MOVE11;
ELSE WHEN E13 THEN MOVE = MOVE13;
ELSE WHEN E31 THEN MOVE = MOVE31;
ELSE WHEN E33 THEN MOVE = MOVE33;
ELSE WHEN E12 THEN MOVE = MOVE12;
ELSE WHEN E21 THEN MOVE = MOVE21;
ELSE WHEN E23 THEN MOVE = MOVE23;
ELSE WHEN E32 THEN MOVE = MOVE32;
ELSE MOVE = NONE;

end pick2
```

ahorrar dos terminales. El método que utilizaremos aquí es combinar las señales para las celdas del borde medio 12, 21, 23 y 32 para producir cuatro señales E12, E21, E23 y E32 que son afirmadas si y sólo si las celdas correspondientes están vacías. Esto puede hacerse con cuatro compuertas NOR de 2 entradas y realmente deja dos entradas o salidas libres en el 22V10.

Suponiendo que las cuatro compuertas NOR son la "otra lógica", la tabla 6-16 proporciona un programa para el PLD PICK2. Cuando se debe seleccionar un movimiento, este programa utiliza la heurística más simple posible: selecciona una celda de la esquina si una está vacía, de lo contrario se elige una celda del extremo medio. Este programa podría emplear alguna mejora, porque algunas veces ganará el jugador (véase el ejercicio 6.8). Por fortuna, las ecuaciones resultantes de la tabla 6-16 requieren solamente de 8 a 10 términos por salida, de modo que se puede aumentar la "inteligencia" (véanse ejercicios 6.9 y 6.10).

6.3 Ejemplos de diseño utilizando VHDL

6.3.1 Desplazado rápido

En la página 468 definimos un desplazador rápido como un circuito lógico combinacional con n entradas de datos, n salidas de datos y un conjunto de entradas de control que especifican cómo desplazar los datos entre la entrada y la salida. Mostramos en la sección 6.1.1 cómo construir un desplazador rápido simple que realiza solamente desplazamientos circulares a la izquierda empleando bloques de construcción MSI. Después, en la sección 6.2.1, presentamos cómo definir un desplazador rápido más capaz utilizando ABEL, pero también señalamos que los PLD normalmente no son adecuados para realizar desplazadores rápidos. En esta subsección demostraremos cómo puede emplearse VHDL para describir tanto el comportamiento como la estructura de los desplazadores rápidos para realización FPGA o ASIC.

La tabla 6-17 es un programa VHDL de comportamiento para un desplazador rápido de 16 bits que efectúa cualquiera de seis diferentes combinaciones de dirección y tipo de desplazamiento. Los tipos de desplazamientos son circulares, lógicos y aritméticos, como se definió anteriormente en la tabla 6-3 y las direcciones son, naturalmente, izquierda y derecha. Como se mostró en la declaración de entidad, una entrada `S` de control de 4 bits proporciona la cantidad de desplazamiento y una entrada de control `C` de 3 bits suministra el modo de desplazamiento (tipo y dirección). Utilizamos el paquete `std_logic_arith` de IEEE y definimos la cantidad de desplazamiento `S` como del tipo `UNSIGNED` de modo que podríamos utilizar después la función `CONV_INTEGER` en ese paquete.

Nótese que la declaración de entidad incluye seis definiciones de constante que establecen la correspondencia entre modos de desplazamiento y el valor de `C`. Aunque no lo discutimos en la sección 4.7, VHDL permite poner constante, tipo, señal y otras declaraciones dentro de una declaración de entidad. Tiene sentido definir tales elementos dentro de la declaración de entidad sólo si deben ser los mismos en cualquier arquitectura. En este caso, estamos sujetando las codificaciones de modo de desplazamiento, de manera que deberían ir aquí. Otros elementos deberían ir en la definición de arquitectura.

En la parte de arquitectura del programa definimos seis funciones, una para cada clase de desplazamiento en un `STD_LOGIC_VECTOR` de 16 bits. Definimos el subtipo `DATAWORD` para ahorrar escritura en las definiciones de función.

Tabla 6-17 Descripción de comportamiento VHDL de un desplazador rápido de 6 funciones.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity barrel16 is
    port (
        DIN: in STD_LOGIC_VECTOR (15 downto 0);  -- Entradas de datos
        S: in UNSIGNED (3 downto 0);             -- Cantidad de desplazamiento, 0-15
        C: in STD_LOGIC_VECTOR (2 downto 0);     -- Control de modo
        DOUT: out STD_LOGIC_VECTOR (15 downto 0) -- Salida de bus de datos
    );
    constant Lrotate:  STD_LOGIC_VECTOR := "000";   -- Defina la codificación de
    constant Rrotate:  STD_LOGIC_VECTOR := "001";   -- los diferentes modos de desplazamiento
    constant Llogical: STD_LOGIC_VECTOR := "010";
    constant Rlogical: STD_LOGIC_VECTOR := "011";
    constant Larith:   STD_LOGIC_VECTOR := "100";
    constant Rarith:   STD_LOGIC_VECTOR := "101";
end barrel16;

architecture barrel16_behavioral of barrel16 is
subtype DATAWORD is STD_LOGIC_VECTOR(15 downto 0);

function Vrol (D: DATAWORD; S: UNSIGNED)
     return DATAWORD is
  variable N: INTEGER;
  variable TMPD: DATAWORD;
  begin
    N := CONV_INTEGER(S); TMPD := D;
    for i in 1 to N loop
      TMPD := TMPD(14 downto 0) & TMPD(15);
    end loop;
    return TMPD;
  end Vrol;
...
begin
process(DIN, S, C)
  begin
    case C is
      when Lrotate  => DOUT <= Vrol(DIN,S);
      when Rrotate  => DOUT <= Vror(DIN,S);
      when Llogical => DOUT <= Vsll(DIN,S);
      when Rlogical => DOUT <= Vsrl(DIN,S);
      when Larith   => DOUT <= Vsla(DIN,S);
      when Rarith   => DOUT <= Vsra(DIN,S);
      when others   => null;
    end case;
  end process;
end barrel16_behavioral;
```

REALICE SUS PROPIOS ROLLOS

VHDL-93 en realidad tiene operadores de arreglo integrados, `rol`, `ror`, `sll`, `srl`, `sla`, y `sra`, correspondientes a las operaciones de desplazamiento que definimos en la tabla 6-3. Puesto que estas operaciones no se incluyen en VHDL-87, hemos definido nuestras propias funciones en la tabla 6-17. Bueno, bueno, en realidad hemos definido sólo una de ellas (`Vrol`); el resto se dejan como un ejercicio para el lector (ejercicio 6.11).

La tabla 6-17 muestra los detalles de sólo la primera función (`Vrol`); el resto son similares con sólo un cambio de una línea. Definimos una variable `N` para convertir la cantidad de desplazamiento `S` en un entero para el ciclo `for`. También asignamos el vector de entrada `D` a una variable local `TMPD`, la cual es desplazada `N` veces en el ciclo `for`. En el cuerpo del ciclo `for`, una instrucción de asignación simple toma un segmento de 15 bits de la palabra de datos (`TMPD 14 downto 0`) y emplea concatenación (`&`) para ponerlo junto con el bit que se sitúa en el extremo izquierdo (`TMPD(15)`). Otros tipos de desplazamiento pueden describirse con operaciones semejantes. Advierta que las funciones de desplazamiento pueden no estar definidas en otras descripciones de la entidad `barrel16`, por ejemplo en arquitecturas estructurales.

La parte de "instrucciones concurrentes" de la arquitectura es un proceso simple que tiene todas las entradas de la entidad en su lista de sensibilidad. Dentro de este proceso, una asignación `case` asigna un resultado a `DOUT` al llamar la función apropiada basada en el valor de la entrada de control de modo `C`.

El proceso en la tabla 6-17 es una bonita descripción de comportamiento del desplazador rápido, pero la mayoría de las herramientas de síntesis no pueden sintetizar un circuito a partir de ella. El problema es que la mayor parte de las herramientas requieren que el intervalo de un ciclo `for` sea estático en el momento en que es analizado. El intervalo del ciclo `for` en la función `Vrol` es dinámico; depende del valor de la señal de entrada cuando el circuito se encuentra en operación.

Bien, eso está bien; es difícil predecir qué clase de circuito presentaría la herramienta de síntesis incluso si pudiera manejar un intervalo `for` dinámico. Éste es un ejemplo donde como diseñadores deberíamos tomar un poco más de control sobre la estructura del circuito para obtener un resultado de síntesis eficiente y razonablemente rápido.

En la figura 6-2 de la página 470 mostramos cómo diseñar un desplazador de barril de 16 bits para desplazamientos circulares a la izquierda utilizando bloques de construcción MSI. Utilizamos una cascada de cuatro multiplexores de 2 entradas y 16 bits para desplazar sus entradas por 0 o 1, 2, 4 u 8 posiciones, dependiendo de los valores de S0 hasta S3, respectivamente. Podemos expresar la misma clase de comportamiento y estructura empleando el programa VHDL mostrado en la tabla 6-18. Aun cuando el programa utilice un proceso de tipo "de comportamiento" en estilo, podemos estar seguros de que la mayoría de los motores de síntesis generarán un multiplexor de una entrada para cada instrucción "`if`" del programa, con lo que se crea una cascada similar a la que se muestra en la figura 6-2.

Tabla 6-18 Programa VHDL para un desplazador en barril de 16 bits para desplazamientos circulares a la izquierda solamente.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity rol16 is
    port (
        DIN: in STD_LOGIC_VECTOR(15 downto 0);  -- Entradas de datos
        S: in STD_LOGIC_VECTOR (3 downto 0);    -- Cantidad de desplazamiento, 0-15
        DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Salida de bus de datos
    );
end rol16;

architecture rol16_arch of rol16 is
begin
process(DIN, S)
  variable X, Y, Z: STD_LOGIC_VECTOR(15 downto 0);
  begin
    if S(0)='1' then X := DIN(14 downto 0) & DIN(15); else X := DIN; end if;
    if S(1)='1' then Y := X(13 downto 0) & X(15 downto 14); else Y := X; end if;
    if S(2)='1' then Z := Y(11 downto 0) & Y(15 downto 12); else Z := Y; end if;
    if S(3)='1' then DOUT <= Z(7 downto 0) & Z(15 downto 8); else DOUT <= Z; end if;
  end process;
end rol16_arch;
```

Naturalmente, el enunciado de nuestro problema requiere un desplazador rápido que pueda desplazarse tanto a la izquierda como a la derecha. La tabla 6-19 modifica el programa anterior para hacer desplazamientos circulares en cualquier dirección. Una entrada adicional, `DIR`, especifica la dirección de desplazamiento, 0 para la izquierda, 1 para la derecha. Cada rango de desplazamiento está especificado por una instrucción `case` que selecciona una de cuatro posibilidades basado en los valores de `DIR` y el bit de `S` que controla ese rango. Advierta que creamos variables de 2 bits locales `CTRLi` para mantener el par de valores `DIR` y `S(i)`; cada instrucción `case` es controlada por una de estas variables. Usted desearía eliminar estas variables y controlar simplemente cada instrucción `case` con una concatenación "`DIR & S (i)`", pero la sintaxis de VHDL no permite eso porque el tipo de esta concatenación sería desconocido.

Una herramienta de síntesis VHDL típica generará un multiplexor de 3 o 4 entradas para cada una de las instrucciones `case` en la tabla 6-19. Una buena herramienta de síntesis generará solamente un multiplexor de 2 entradas para la última instrucción `case`.

De este modo, ahora tenemos un desplazador rápido que hará desplazamientos circulares a la izquierda o a la derecha, pero no hemos terminado todavía: necesitamos tener cuidado de los desplazamientos lógico y aritmético en ambas direcciones. En la figura 6-14 se presenta nuestra estrategia para terminar el diseño. Comenzamos con el componente `ROLR16` que acabamos de completar, y utilizamos otra lógica para controlar la dirección de desplazamiento como una función de `C`.

Tabla 6-19 Programa VHDL para un desplazador rápido de 16 bits para desplazamientos circulares a la izquierda y a la derecha.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity rolr16 is
    port (
        DIN:  in STD_LOGIC_VECTOR(15 downto 0);  -- Entradas de datos
        S:    in STD_LOGIC_VECTOR (3 downto 0);  -- Cantidad de desplazamiento, 0-15
        DIR:  in STD_LOGIC;                      -- Dirección de desplazamiento,
                                                    0=>L, 1=>R
        DOUT: out STD_LOGIC_VECTOR(15 downto 0)  -- Salida de bus de datos
    );
end rolr16;

architecture rol16r_arch of rolr16 is
begin
process(DIN, S, DIR)
  variable X, Y, Z: STD_LOGIC_VECTOR(15 downto 0);
  variable CTRL0, CTRL1, CTRL2, CTRL3: STD_LOGIC_VECTOR(1 downto 0);
  begin
    CTRL0 := S(0) & DIR; CTRL1 := S(1) & DIR; CTRL2 := S(2) & DIR; CTRL3 := S(3) & DIR;
    case CTRL0 is
      when "00" | "01" => X := DIN;
      when "10" => X := DIN(14 downto 0) & DIN(15);
      when "11" => X := DIN(0) & DIN(15 downto 1);
      when others => null;  end case;
    case CTRL1 is
      when "00" | "01" => Y := X;
      when "10" => Y := X(13 downto 0) & X(15 downto 14);
      when "11" => Y := X(1 downto 0) & X(15 downto 2);
      when others => null;  end case;
    case CTRL2 is
      when "00" | "01" => Z := Y;
      when "10" => Z := Y(11 downto 0) & Y(15 downto 12);
      when "11" => Z := Y(3 downto 0) & Y(15 downto 4);
      when others => null;  end case;
    case CTRL3 is
      when "00" | "01" => DOUT <= Z;
      when "10" | "11" => DOUT <= Z(7 downto 0) & Z(15 downto 8);
      when others => null;  end case;
  end process;
end rol16r_arch;
```

A continuación debemos "arreglar" alguno de los bits de resultado si estamos haciendo un desplazamiento lógico o aritmético. Para un desplazamiento lógico o aritmético de n bits a la izquierda, es necesario establecer los $n - 1$ bits más a la derecha a 0 o al valor del bit más a la derecha original, respectivamente. Para obtener un desplazamiento lógico o aritmético de n bits a la derecha, debemos establecer los $n - 1$ bits más a la derecha a 0 o al valor del bit más a la derecha original, respectivamente.

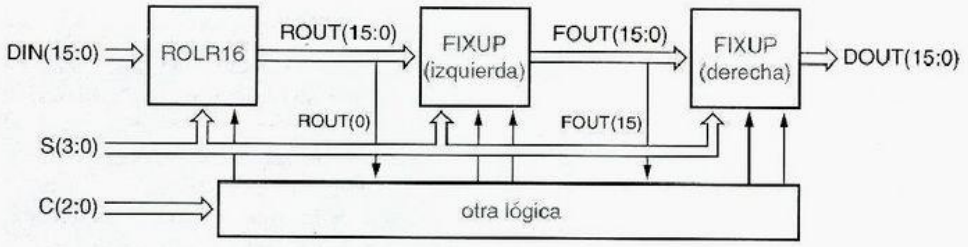


Figura 6-14 Componentes del desplazador rápido.

Como se ilustra en la figura 6-14, nuestra estrategia es seguir el desplazamiento circular (ROLR16) con un circuito de acondicionamiento (FIXUP) que conecta bits de menor orden apropiados para un desplazamiento lógico o aritmético a la izquierda, y sigue con otro circuito de acondicionamiento que conecta los bits de mayor orden para un desplazamiento lógico o aritmético a la derecha.

La tabla 6-20 es un programa VHDL de comportamiento para el circuito de acondicionamiento de desplazamiento izquierdo. El circuito tiene 16 bits de entrada y salida de datos, `DIN` y `DOUT`. Sus entradas de control son la cantidad de desplazamiento `S`, una entrada de habilitación `FEN` y el nuevo valor `FDAT` que se conectará en los bits de datos acondicionados. Para cada bit de salida `DOUT(i)`, el circuito extrae el valor del bit acondicionado —si `i` es menor que `S`— y el circuito estará habilitado; si no, extrae la entrada de datos sin modificar `DIN(i)`.

El ciclo `for` en la tabla 6-20 es fácilmente sintetizable, pero usted no puede estar seguro de qué clase de lógica generará la herramienta de síntesis. En particular, la operación

Tabla 6-20 Programa VHDL de comportamiento para acondicionamientos de desplazamiento izquierdo.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity fixup is
    port (
        DIN:  in STD_LOGIC_VECTOR(15 downto 0); -- Entradas de datos
        S:    in UNSIGNED(3 downto 0);          -- Cantidad de desplazamiento, 0-15
        FEN:  in STD_LOGIC;                     -- Habilitación de acondicionamiento
        FDAT: in STD_LOGIC;                     -- Datos de acondicionamiento
        DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Salida de bus de datos
    );
end fixup;

architecture fixup_arch of fixup is
begin
process(DIN, S, FEN, FDAT)
  begin
    for i in 0 to 15 loop
      if (i < CONV_INTEGER(S)) and (FEN = '1') then DOUT(i) <= FDAT;
      else DOUT(i) <= DIN(i);  end if;
    end loop;
  end process;
end fixup_arch;
```

UNA ESTRUCTURA EN SERIE DE ACONDICIONAMIENTO (FIX UP)

Una arquitectura estructural para la lógica de acondicionamiento se muestra en la tabla 6-21. Aquí, hemos definido lo que es en efecto un circuito iterativo para crear un vector de 16 bits FSEL, donde FSEL(i) es 1 si el bit i necesita acondicionamiento. Comenzamos por establecer FSEL(15) a 0, puesto que ese bit nunca necesita acondicionarse. Entonces notamos que para los valores restantes de i, FSEL(i) debería ser 1 si S es igual a i+1 o si FSEL(i+1) ya está afirmado. De este modo, la asignación de FSEL dentro de la proposición de generación crea una cadena serial de compuertas OR de 2 entradas, donde una entrada es afirmada si S=i (decodificado con una compuerta AND de 4 entradas) y la otra entrada está conectada a la anterior salida de la compuerta OR. La proposición de asignación DOUT(i) crea 16 multiplexores de 2 entradas que seleccionan ya sea DIN(i) o los datos de acondicionamiento (FDAT), dependiendo del valor de FSEL(i).

Aunque la realización en serie es compacta, es muy lenta en comparación con la que realiza cada salida FSEL como un circuito de suma de productos de 2 niveles. Sin embargo, el retardo extenso puede no importar, porque el circuito de acondicionamiento aparece cerca del final de la trayectoria de datos. Si la velocidad todavía es un problema, hay un truco de cero costo que corta el retardo a la mitad (véase el ejercicio 6.12).

">" en cada paso del ciclo puede causar la síntesis de un comparador de magnitud de propósito general, aun cuando uno de los operandos sea una constante y cada salida podría ser generada, por tanto, con no más que un puñado de compuertas. (¡De hecho, la lógica para "7 < CONV_INTEGER(S)" es precisamente un alambre, S(3)!) Para una versión estructural de esta función, véase el cuadro en esta página.

Para desplazamientos a la derecha, se inicia el acondicionamiento desde el extremo opuesto de la palabra de datos, de modo que parecería que necesitamos una segunda versión del circuito de acondicionamiento. Sin embargo, podemos utilizar la versión original si sólo invertimos el orden de sus bits de entrada y salida, como veremos pronto.

La tabla 6-22 pone juntos una arquitectura estructural para el desplazador rápido completo de 16 bits y 6 funciones utilizando el enfoque de diseño de la figura 6-14 en la página 505. La declaración de entidad para barrel16 no sufre cambios del original en la tabla 6-17. La arquitectura declara dos componentes, rolr16 y fixup; éstos emplean nuestras anteriores definiciones de entidad. La parte de instrucciones de la

Tabla 6-21 Arquitectura VHDL estructural para acondicionamientos de desplazamiento izquierdo.

```
architecture fixup_struc of fixup is
signal FSEL: STD_LOGIC_VECTOR(15 downto 0);       -- Selección de acondicionamiento
begin
  FSEL(15) <= '0'; DOUT(15) <= DIN(15);
  U1: for i in 14 downto 0 generate
    FSEL(i) <= '1' when CONV_INTEGER(S) = i+1 else FSEL(i+1);
    DOUT(i) <= FDAT when (FSEL(i) = '1' and FEN = '1') else DIN(i);
  end generate;
end fixup_struc;
```

arquitectura crea instancias de `rolr16` y `fixup` y tiene diversas instrucciones de asignación que crean señales de control necesarias (la "otra lógica" en la figura 6-14).

Por ejemplo, la primera instrucción de asignación afirma `DIR_RIGHT` si `C` especifica uno de los desplazamientos a la derecha. Las entradas de habilitación para los circuitos de acondicionamiento izquierdo y derecho son `FIX_LEFT` y `FIX_RIGHT`, y están afirmados por desplazamientos lógicos y aritméticos izquierdo y derecho. Los valores de datos acondicionados son `FIX_LEFT_DAT` y `FIX_RIGHT_DAT`.

Mientras que todas las instrucciones en la arquitectura se ejecutan de manera concurrente, se numeran en la tabla 6-22 en el orden del flujo de datos real para mejorar la legibilidad. Primero, `rolr16` es instanciado para realizar el desplazamiento circular básico a la izquierda o a la derecha, como se especifique. Sus salidas están conectadas a las entradas del primer componente `fixup` (U2) para manipular acondicionamientos para desplazamientos lógicos y aritméticos a la izquierda. A continuación viene U3, una instruc-

Tabla 6-22 Arquitectura estructural VHDL para el desplazador rápido de 6 funciones.

```
architecture barrel16_struc of barrel16 is

component rolr16 port (
        DIN:  in STD_LOGIC_VECTOR(15 downto 0); -- Entradas de datos
        S:    in UNSIGNED(3 downto 0);          -- Cantidad de desplazamiento, 0-15
        DIR:  in STD_LOGIC;                     -- Dirección de desplazamiento,
                                                   0=>L, 1=>R
        DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Salida de bus de datos
    );  end component;

component fixup port (
        DIN:  in STD_LOGIC_VECTOR(15 downto 0); -- Entradas de datos
        S:    in UNSIGNED(3 downto 0);          -- Cantidad de desplazamiento, 0-15
        FEN:  in STD_LOGIC;                     -- Habilitación de acondicionamiento
        FDAT: in STD_LOGIC;                     -- Datos de acondicionamiento
        DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Salida de bus de datos
    );  end component;

signal DIR_RIGHT, FIX_RIGHT, FIX_RIGHT_DAT, FIX_LEFT, FIX_LEFT_DAT: STD_LOGIC;
signal ROUT, FOUT, RFIXIN, RFIXOUT: STD_LOGIC_VECTOR(15 downto 0);

begin
  DIR_RIGHT <= '1' when C = Rrotate or C = Rlogical or C = Rarith else '0';
  FIX_LEFT <= '1' when DIR_RIGHT='0' and (C = Llogical or C = Larith) else '0';
  FIX_RIGHT <= '1' when DIR_RIGHT='1' and (C = Rlogical or C = Rarith) else '0';
  FIX_LEFT_DAT <= DIN(0) when C = Larith else '0';
  FIX_RIGHT_DAT <= DIN(15) when C = Rarith else '0';
  U1: rolr16 port map (DIN, S, DIR_RIGHT, ROUT);
  U2: fixup port map (ROUT, S, FIX_LEFT, FIX_LEFT_DAT, FOUT);
  U3: for i in 0 to 15 generate RFIXIN(i) <= FOUT(15-i); end generate;
  U4: fixup port map (RFIXIN, S, FIX_RIGHT, FIX_RIGHT_DAT, RFIXOUT);
  U5: for i in 0 to 15 generate DOUT(i) <= RFIXOUT(15-i); end generate;
end barrel16_struc;
```

ESTILO DE OCULTAMIENTO DE INFORMACIÓN

Basado en la codificación de C, usted podría querer reemplazar la primera proposición de asignación en la tabla 6-22 con "`DIR_RIGHT <= C(0)`", lo que garantizaría llevar a una realización más eficaz para ese bit de control: ¡sólo un alambre! Sin embargo, esto violaría un principio de programación de ocultamiento de información y conduciría a posibles errores.

Escribimos de manera explícita las codificaciones de desplazamiento utilizando definiciones `constant` en la declaración de entidad `barrel16`. La arquitectura no necesita ser prevenida de los detalles de la codificación. Supongamos que, no obstante, hacemos el cambio de arquitectura sugerido arriba. Si alguien más (¡o nosotros mismos!) llegamos después de un tiempo y cambiamos las definiciones `constant` en la entidad `barrel16` para hacer una codificación diferente, ¡la arquitectura no utilizaría las nuevas codificaciones! El ejercicio 6.13 le solicitará a usted cambiar las definiciones de modo que los ahorros de costo de nuestro cambio sugerido sean habilitados por la definición de entidad.

ción `generate` que invierte el orden de las entradas de datos para el siguiente componente `fixup` (`U4`), que maneja los acondicionamientos para desplazamientos lógicos y aritméticos a la derecha. Finalmente `U5`, otra instrucción `generate`, deshace la inversión de bits de `U3`. Advierta que en síntesis, `U3` y `U5` son meramente permutaciones de alambres.

Muchas otras arquitecturas son posibles para la entidad original `barrel16`. En el ejercicio 6.14, sugerimos una arquitectura que habilita el desplazamiento circular para hacerce mediante la entidad `rol16`, la que utiliza solamente multiplexores de 2 entradas, en vez de la más costosa `rol16`.

6.3.2 Codificador de punto flotante simple

Definimos un formato de número de punto flotante simple en la página 471 y planteamos el problema de diseño de convertir un número del formato de punto fijo al de punto flotante. El problema de determinar el exponente del número de punto flotante se mapeaba bien en un codificador de prioridad MSI. En un HDL, el mismo problema se mapea en instrucciones anidadas "`if`".

La tabla 6-23 es un programa de comportamiento para el codificador de punto flotante. Dentro de la arquitectura `fpenc_arch`, una instrucción anidada "`if`" verifica el intervalo de la entrada `B` y establece `M` y `E` de manera apropiada. Nótese que el programa hace uso del paquete IEEE `std_logic_arith`; esto se hace para obtener el tipo `UNSIGNED` y las operaciones de comparación que van junto con él, como describimos en la sección 5.9.6. Sólo para ahorrar texto, se define una variable `BU` para mantener el valor de B a medida que es convertido al tipo `UNSIGNED`; alternativamente, podíamos haber escrito "`UNSIGNED(B)`" en cada cláusula anidada "`if`".

Aunque el código en la tabla 6-23 es completamente sintetizable, algunas herramientas de síntesis pueden no ser lo suficientemente inteligentes para reconocer que las comparaciones anidadas requieren sólo un bit para ser verificadas en cada nivel, y pueden en su lugar generar un comparador completo de 11 bits en cada nivel. Tal lógica sería mucho más grande y lenta que lo que, de otro modo, sería posible. Si nos enfrentamos con este problema, siempre podemos escribir la arquitectura de un modo un poco distinto y más explícitamente para ayudar a la herramienta, como se muestra en la tabla 6-24.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity fpenc is
  port (
    B: in STD_LOGIC_VECTOR(10 downto 0); -- número de punto fijo
    M: out STD_LOGIC_VECTOR(3 downto 0); -- mantisa de punto flotante
    E: out STD_LOGIC_VECTOR(2 downto 0)  -- exponente de punto flotante
  );
end fpenc;

architecture fpenc_arch of fpenc is
begin
  process(B)
  variable BU: UNSIGNED(10 downto 0);
  begin
    BU := UNSIGNED(B);
    if    BU < 16   then M <= B( 3 downto 0); E <= "000";
    elsif BU < 32   then M <= B( 4 downto 1); E <= "001";
    elsif BU < 64   then M <= B( 5 downto 2); E <= "010";
    elsif BU < 128  then M <= B( 6 downto 3); E <= "011";
    elsif BU < 256  then M <= B( 7 downto 4); E <= "100";
    elsif BU < 512  then M <= B( 8 downto 5); E <= "101";
    elsif BU < 1024 then M <= B( 9 downto 6); E <= "110";
    else                 M <= B(10 downto 7); E <= "111";
    end if;
  end process;
end fpenc_arch;
```

Tabla 6-23 Programa VHDL de comportamiento para conversión de punto fijo a punto flotante.

```
architecture fpence_arch of fpenc is
begin
  process(B)
  begin
    if   B(10) = '1' then M <= B(10 downto 7); E <= "111";
    elsif B(9) = '1' then M <= B( 9 downto 6); E <= "110";
    elsif B(8) = '1' then M <= B( 8 downto 5); E <= "101";
    elsif B(7) = '1' then M <= B( 7 downto 4); E <= "100";
    elsif B(6) = '1' then M <= B( 6 downto 3); E <= "011";
    elsif B(5) = '1' then M <= B( 5 downto 2); E <= "010";
    elsif B(4) = '1' then M <= B( 4 downto 1); E <= "001";
    else                  M <= B( 3 downto 0); E <= "000";
    end if;
  end process;
end fpence_arch;
```

Tabla 6-24 Arquitectura VHDL alternativa para la conversión de punto fijo a punto flotante.

LA B NO ES MI TIPO

En la tabla 6-23 empleamos la expresión `UNSIGNED(B)` para convertir `B`, en un arreglo de tipo `STD_LOGIC_VECTOR` en un arreglo de tipo `UNSIGNED`. Esto se denomina una *conversión explícita de tipo*. VHDL nos permite convertir entre tipos cercanamente relacionados al escribir el tipo deseado seguido por el valor, que será convertido con paréntesis. Dos tipos arreglo están "cercanamente relacionados" si tienen el mismo tipo de elemento, el mismo número de dimensiones y los mismos tipos índice (típicamente `INTEGER`) o los que pueden ser convertidos al tipo. Los valores en el arreglo antiguo se colocan en las posiciones correspondientes, de izquierda a derecha, en el nuevo arreglo.

Por otro lado, podemos utilizar los comparadores reales y dedicar aún más compuertas para mejorar la funcionalidad de nuestro diseño. En particular, el presente diseño realiza un truncamiento en vez de un redondeo cuando se generan los bits de la mantisa. Un resultado más preciso se consigue cuando se redondea, pero éste es un diseño mucho más complicado. Primero, necesitaríamos un sumador para sumar 1 a los bits seleccionados de la mantisa cuando redondeáramos. Sin embargo, sumar 1 cuando la mantisa ya es 1111 nos proyectaría al siguiente intervalo de exponente, de modo que necesitaríamos tener precaución en este caso. Finalmente, nunca podremos redondear si la mantisa sin redondear y el exponente son 1111 y 111, porque no hay valor superior en nuestra representación de punto flotante para redondear.

El programa en la tabla 6-25 realiza el redondeo como se desea. La función `round` toma un segmento seleccionado de 5 bits del número de punto fijo y devuelve los cuatro bits de mayor orden, sumando 1 si el LSB es 1. Por lo tanto, si pensamos en el punto binario como situado justo a la izquierda del LSB, el redondeo ocurre si la parte truncada de la mantisa es 1/2 o más. En cada cláusula de la instrucción anidada "`if`" del proceso, el valor de comparación se selecciona de manera que se presente redondeo hacia arriba solamente si no hay desbordamiento, llevando el resultado al siguiente intervalo del exponente. De otro modo, ocurren la conversión y el redondeo en la siguiente cláusula. En la última cláusula, nos aseguramos que no se redondee hacia arriba cuando estemos al final del intervalo de punto flotante.

ENGULLIR, ENGULLIR

La operación de redondeo no requiere un sumador de 4 bits, solamente un "incrementador", puesto que uno de los sumandos siempre es 1. Algunas herramientas de VHDL pueden sintetizar el sumador completo, mientras que otras pueden ser lo suficientemente inteligentes para emplear un incrementador con muy pocas compuertas.

En algunos casos puede no importar. Las herramientas más sofisticadas para diseño FPGA y ASIC incluyen *engullidores de compuerta*. Estos programas rastrean compuertas con entradas constantes, y como resultado eliminan compuertas o entradas de compuertas. Por ejemplo, una entrada de compuerta AND con un 1 constante aplicado a ella puede ser eliminada y una compuerta AND con una entrada 0 constante puede ser reemplazada con una señal constante de 0.

Un programa engullidor de compuertas propaga los efectos de las entradas constantes tanto como es posible en un circuito. De este modo, puede transformar un sumador de 4 bits con una entrada constante de 1 en un incrementador más económico de 4 bits.

Tabla 6-25
Arquitectura VHDL de comportamiento para la conversión de punto fijo a punto flotante con redondeo.

```
architecture fpencr_arch of fpenc is
function round (BSLICE: STD_LOGIC_VECTOR(4 downto 0))
    return STD_LOGIC_VECTOR is
  variable BSU: UNSIGNED(3 downto 0);
  begin
    if BSLICE(0) = '0' then return BSLICE(4 downto 1);
    else null;
      BSU := UNSIGNED(BSLICE(4 downto 1)) + 1;
      return STD_LOGIC_VECTOR(BSU);
    end if;
  end;
begin
  process(B)
  variable BU: UNSIGNED(10 downto 0);
  begin
    BU := UNSIGNED(B);
    if    BU < 16       then M <= B( 3 downto 0); E <= "000";
    elsif BU < 32-1     then   <= round(B( 4 downto 0)); E <= "001";
    elsif BU < 64-2     then M <= round(B( 5 downto 1)); E <= "010";
    elsif BU < 128-4    then M <= round(B( 6 downto 2)); E <= "011";
    elsif BU < 256-8    then M <= round(B( 7 downto 3)); E <= "100";
    elsif BU < 512-16   then M <= round(B( 8 downto 4)); E <= "101";
    elsif BU < 1024-32  then M <= round(B( 9 downto 5)); E <= "110";
    elsif BU < 2048-64  then M <= round(B(10 downto 6)); E <= "111";
    else                     M <= "1111";                E <= "111";
    end if;
  end process;
end fpencr_arch;
```

Una vez más, los resultados de la síntesis para este programa de comportamiento pueden o no ser eficaces. Además de las múltiples instrucciones de comparación, ahora debemos preocuparnos acerca de los múltiples sumadores de 4 bits que pueden ser sintetizados como resultado de las múltiples llamadas a la función `round`. En el ejercicio (6.15) se debe reestructurar la arquitectura de modo que solamente se sintetice un sumador simple.

6.3.3 Codificador de prioridad dual

En este ejemplo utilizaremos VHDL para crear una descripción de comportamiento de un codificador de prioridad PLD que identifica tanto la señal afirmada de más alta prioridad como la de segunda más alta prioridad entre un conjunto de entradas de solicitud `R (0 a 7)`, donde `R(0)` tiene la prioridad mayor. Haremos uso de `A(2 downto 0)` y `AVALID` para identificar la solicitud de más alta prioridad, poniendo a 1 `AVALID` solamente si se presenta una solicitud de más alta prioridad. De manera similar, utilizaremos `B(2 downto 0)` y `BVALID` para identificar la solicitud de segunda más alta prioridad.

Tabla 6-26
Programa VHDL de comportamiento para un codificador de prioridad dual.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity Vprior2 is
    port (
        R: in STD_LOGIC_VECTOR (0 to 7);
        A, B: out STD_LOGIC_VECTOR (2 downto 0);
        AVALID, BVALID: buffer STD_LOGIC
    );
end Vprior2;

architecture Vprior2_arch of Vprior2 is
begin
  process(R, AVALID, BVALID)
  begin
    AVALID <= '0'; BVALID <= '0'; A <= "000"; B <= "000";
    for i in 0 to 7 loop
      if R(i) = '1' and AVALID = '0' then
        A <= CONV_STD_LOGIC_VECTOR(i,3); AVALID <= '1';
      elsif R(i) = '1' and BVALID = '0' then
        B <= CONV_STD_LOGIC_VECTOR(i,3); BVALID <= '1';
      end if;
    end loop;
  end process;
end Vprior2_arch;
```

La tabla 6-26 muestra un programa de comportamiento VHDL para el codificador de prioridad. En lugar del enfoque de "`if`" anidado del ejemplo anterior, haremos uso de un ciclo "`for`". Este enfoque nos permitirá tener cuidado tanto de la primera como de la segunda prioridad dentro del mismo ciclo, trabajando de la más alta hasta la más baja prioridad. Además de `std_logic_1164`, el programa utiliza el paquete `std_logic_arith` IEEE a fin de conseguir la función `CONV_STD_LOGIC_VECTOR`. También escribimos esta función explícitamente en la tabla 4-39 de la página 279.

Como podemos ver en la tabla, los puertos `AVALID` y `BVALID` están declarados como de modo `buffer`, porque se leen dentro de la arquitectura. Si estuviéramos atorados con una definición de entidad que declarara `AVALID` y `BVALID` en modo `out`, podríamos aún utilizar el mismo enfoque de arquitectura, pero se tendrían que declarar variables locales correspondientes a `AVALID` y `BVALID` dentro del proceso. Advierta también que incluimos `AVALID` y `BVALID` en la lista de sensibilidad del proceso. Aunque esto no es estrictamente necesario, evita advertencias que el compilador daría de otro modo acerca del uso del valor de una señal que no está en la lista de sensibilidad.

El enfoque del "`if`" anidado también puede ser utilizado para el codificador de prioridad dual, pero produce un programa más grande con más probabilidad de que ocurran accidentes, como se muestra en la tabla 6-27. Por otra parte, puede producir un mejor resultado de síntesis; la única manera de saber con una herramienta en particular

```
architecture Vprior2i_arch of Vprior2 is
begin
  process(R, A, AVALID, BVALID)
  begin
    if    R(0) = '1' then A <= "000"; AVALID <= '1';
    elsif R(1) = '1' then A <= "001"; AVALID <= '1';
    elsif R(2) = '1' then A <= "010"; AVALID <= '1';
    elsif R(3) = '1' then A <= "011"; AVALID <= '1';
    elsif R(4) = '1' then A <= "100"; AVALID <= '1';
    elsif R(5) = '1' then A <= "101"; AVALID <= '1';
    elsif R(6) = '1' then A <= "110"; AVALID <= '1';
    elsif R(7) = '1' then A <= "111"; AVALID <= '1';
    else                  A <= "000"; AVALID <= '0';
    end if;
    if    R(1) = '1' and A /= "001" then B <= "001"; BVALID <= '1';
    elsif R(2) = '1' and A /= "010" then B <= "010"; BVALID <= '1';
    elsif R(3) = '1' and A /= "011" then B <= "011"; BVALID <= '1';
    elsif R(4) = '1' and A /= "100" then B <= "100"; BVALID <= '1';
    elsif R(5) = '1' and A /= "101" then B <= "101"; BVALID <= '1';
    elsif R(6) = '1' and A /= "110" then B <= "110"; BVALID <= '1';
    elsif R(7) = '1' and A /= "111" then B <= "111"; BVALID <= '1';
    else                                 B <= "000"; BVALID <= '0';
    end if;
  end process;
end Vprior2i_arch;
```

Tabla 6-27 Arquitectura VHDL alternativa para un codificador de prioridad dual.

es sintetizar el circuito y analizar los resultados en términos del retardo y conteo de compuerta o celda.

Tanto las instrucciones anidadas "`if`" como las instrucciones "`for`" pueden conducir a largas cadenas de retardo en síntesis. Para garantizar que se obtendrá un codificador de prioridad dual más rápido, debe seguir un enfoque de diseño estructural o semiestructural. Por ejemplo, se puede comenzar por escribir un modelo de flujo de datos de un codificador de prioridad de 8 entradas rápido utilizando las ideas encontradas en el diagrama lógico 74x148 (figura 5-50 de la página 379) o en un programa ABEL relacionado (tabla 5-24 de la página 382). Entonces se pueden poner dos de estas juntas en una estructura que "deje fuera de combate" la entrada de más alta prioridad a fin de encontrar la segunda, como mostramos en la figura 6-6 de la página 475.

6.3.4 Comparadores en cascada

Los comparadores en cascada son algo que normalmente no haríamos en un modelo de comportamiento VHDL, porque el lenguaje y el paquete `std_logic_arith` IEEE definen comparadores de cualquier longitud deseada directamente. Sin embargo, podemos necesitar en verdad escribir programas VHDL estructural o semiestructural que conecten componentes más pequeños de comparador en una manera específica para obtener un rendimiento elevado.

Tabla 6-28
Programa VHDL de comportamiento para un comparador de 64 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity comp64 is
    port ( A, B: in STD_LOGIC_VECTOR (63 downto 0);
           EQ, GT: out STD_LOGIC );
end comp64;

architecture comp64_arch of comp64 is
begin
  EQ <= '1' when A = B else '0';
  GT <= '1' when A > B else '0';
end comp64_arch;
```

La tabla 6-28 es un modelo de comportamiento simple de un comparador de 64 bits con salidas igual y mayor que. Este programa utiliza el paquete `std_logic_unsigned` IEEE, cuyas funciones de comparación integradas tratan todas las señales del tipo `STD_LOGIC_VECTOR` como enteros sin signo. Aunque el programa es completamente sintetizable, la velocidad y el tamaño del resultado dependen de la "inteligencia" de la herramienta particular que se utiliza.

Una alternativa es construir el comparador mediante componentes más pequeños en cascada, tales como los comparadores de 8 bits. La tabla 6-29 es el modelo de comportamiento de un comparador de 8 bits. Una herramienta particular puede o no puede sintetizar un comparador muy rápido a partir de este programa, pero es seguro que sea significativamente más rápido que un comparador de 64 bits en cualquier caso.

A continuación, podemos escribir un programa estructural que instancie ocho de estos comparadores de 8 bits y conecte sus salidas individuales a través de lógica adicional para calcular el resultado de comparación total. Una manera de hacer esto se muestra en la tabla 6-30. Una instrucción `generate` crea no sólo los comparadores de 8 bits

Tabla 6-29
Programa VHDL para un comparador de 8 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity comp8 is
    port ( A, B: in STD_LOGIC_VECTOR (7 downto 0);
           EQ, GT: out STD_LOGIC );
end comp8;

architecture comp8_arch of comp8 is
begin
  EQ <= '1' when A = B else '0';
  GT <= '1' when A > B else '0';
end comp8_arch;
```

Tabla 6-30 Arquitectura estructural VHDL para un comparador de 64 bits.

```
architecture comp64s_arch of comp64 is
component comp8
  port ( A, B: in STD_LOGIC_VECTOR (7 downto 0);
         EQ, GT: out STD_LOGIC);
end component;
signal EQ8, GT8: STD_LOGIC_VECTOR (7 downto 0); -- =, > para segmento de 8 bits
signal SEQ, SGT: STD_LOGIC_VECTOR (8 downto 0); -- cadena en serie de resultados de segmento
begin
  SEQ(8) <= '1'; SGT(8) <= '0';
  U1: for i in 7 downto 0 generate
        U2: comp8 port map (A(7+i*8 downto i*8), B(7+i*8 downto i*8), EQ8(i), GT8(i));
        SEQ(i) <= SEQ(i+1) and EQ8(i);
        SGT(i) <= SGT(i+1) or (SEQ(i+1) and GT8(i));
  end generate;
  EQ <= SEQ(0); GT <= SGT(0);
end comp64s_arch;
```

individuales, sino también lógica en cascada que construye en serie el resultado total de la etapa más significativa a la menos significativa.

Una herramienta no compleja podría sintetizar un circuito comparador iterativo lento para nuestra arquitectura de comparador de 64 bits original en la tabla 6-28. En esta situación, la arquitectura en la tabla 6-30 produce un circuito sintetizado más rápido porque "empuja" de manera explícita la información en cascada para cada segmento de 8 bits y los combina en un circuito combinacional más rápido (apenas 8 niveles de lógica AND-OR, no 64). Una herramienta más sofisticada puede allanar el comparador de 8 bits a una estructura no iterativa más rápida, similar al comparador MSI 74x682 (la figura 5-84 en la página 425), y puede allanar nuestra lógica en cascada iterativa en la tabla 6-30 en ecuaciones de suma de productos de dos niveles semejantes a las de la solución ABEL de la página 487.

6.3.5 Comparador dependiente del modo

Para el ejemplo siguiente, supongamos que tenemos un sistema en el cual necesitamos comparar dos palabras de 32 bits en circunstancias normales, pero en ciertas ocasiones debemos ignorar uno o dos bits de menor orden de las palabras de entrada. El modo de operación es especificado por los dos bits de control de modo, M1 y M0, como se muestra en la tabla 6-9 de la página 488.

La funcionalidad deseada puede obtenerse muy fácilmente en VHDL utilizando una instrucción `case` para seleccionar el comportamiento por modo, como se observa en el programa de la tabla 6-31. Éste es una descripción de comportamiento perfectamente buena y que es sintetizable por completo. Sin embargo, tiene una desventaja principal en la síntesis: con toda seguridad, causará la creación de tres comparadores de magnitud e igualdad separados (32, 31 y 30 bits), uno para cada caso en la instrucción `case`. Los comparadores individuales pueden ser rápidos o no, como se discutió en

Tabla 6-31 Arquitectura de comportamiento VHDL de un comparador dependiente del modo de 32 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Vmodecmp is
    port ( M: in STD_LOGIC_VECTOR (1 downto 0);      -- modo
           A, B: in STD_LOGIC_VECTOR (31 downto 0); -- enteros sin signo
           EQ, GT: out STD_LOGIC );                  -- resultados de comparación
end Vmodecmp;

architecture Vmodecmp_arch of Vmodecmp is
begin
  process (M, A, B)
  begin
    case M is
      when "00" =>
        if A = B then EQ <= '1'; else EQ <= '0'; end if;
        if A > B then GT <= '1'; else GT <= '0'; end if;
      when "01" =>
        if A(31 downto 1) = B(31 downto 1) then EQ <= '1'; else EQ <= '0'; end if;
        if A(31 downto 1) > B(31 downto 1) then GT <= '1'; else GT <= '0'; end if;
      when "10" =>
        if A(31 downto 2) = B(31 downto 2) then EQ <= '1'; else EQ <= '0'; end if;
        if A(31 downto 2) > B(31 downto 2) then GT <= '1'; else GT <= '0'; end if;
      when others => EQ <= '0';  GT <= '0';
    end case;
  end process;
end Vmodecmp_arch;
```

la subsección anterior, pero no nos preocuparemos acerca de la velocidad para este ejemplo.

Una alternativa más eficiente es realizar sólo una comparación para los 30 bits de orden alto de las entradas, utilizando lógica adicional que es dependiente del modo para dar un resultado final empleando los bits de orden bajo como sea necesario. Este enfoque se muestra en la tabla 6-32. Dos variables, `EQ30` y `GT30`, se utilizan dentro del proceso para mantener los resultados de la comparación de los 30 bits de orden alto. Una instrucción `case` semejante a la de la arquitectura anterior se utiliza entonces para obtener los resultados finales como una función del modo. Si se desea, la velocidad de la comparación de 30 bits puede optimizarse utilizando los métodos que se analizaron en la subsección anterior.

6.3.6 Contador de unos

Varios algoritmos importantes incluyen el paso de contar el número de bits "1" en una palabra de datos. De hecho, en fechas recientes, algunos conjuntos de instrucciones para microprocesador se han extendido para incluir el conteo de unos como una instrucción básica. En este ejemplo, supondremos que se nos pide diseñar un circuito combinacional que cuente unos en una palabra de 32 bits como parte de la unidad lógica y aritmética de un microprocesador.

Tabla 6-32 Arquitectura más eficiente para un comparador dependiente del modo de 32 bits.

```
architecture Vmodecpe_arch of Vmodecmp is
begin
  process (M, A, B)
  variable EQ30, GT30: STD_LOGIC; -- resultados de comparación de 30 bits
  begin
    if A(31 downto 2) = B(31 downto 2) then EQ30 := '1'; else EQ30 := '0'; end if;
    if A(31 downto 2) > B(31 downto 2) then GT30 := '1'; else GT30 := '0'; end if;
    case M is
      when "00" =>
        if EQ30='1' and A(1 downto 0) = B(1 downto 0) then
          EQ <= '1'; else EQ <= '0'; end if;
        if GT30='1' or (EQ30='1' and A(1 downto 0) > B(1 downto 0)) then
          GT <= '1'; else GT <= '0'; end if;
      when "01" =>
        if EQ30='1' and A(1) = B(1) then EQ <= '1'; else EQ <= '0'; end if;
        if GT30='1' or (EQ30='1' and A(1) > B(1)) then
          GT <= '1'; else GT <= '0'; end if;
      when "10" =>  EQ <= EQ30;  GT <= GT30;
      when others => EQ <= '0';  GT <= '0';
    end case;
  end process;
end Vmodecpe_arch;
```

El conteo de unos puede describirse con mucha facilidad mediante un programa de comportamiento VHDL, como se muestra en la tabla 6-33. Este programa es completamente sintetizable, pero puede generar una realización muy lenta y poco eficiente con 32 sumadores de 5 bits que se conectan en serie.

Tabla 6-33 Programa VHDL de comportamiento para un contador de unos de 32 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Vcntls is
    port ( D: in STD_LOGIC_VECTOR (31 downto 0);
           SUM: out STD_LOGIC_VECTOR (5 downto 0) );
end Vcntls;

architecture Vcntls_arch of Vcntls is
begin
  process (D)
  variable S: STD_LOGIC_VECTOR(5 downto 0);
  begin
    S:= "000000";
    for i in 0 to 31 loop
      if D(i) = '1' then S := S + "000001"; end if;
    end loop;
    SUM <= S;
  end process;
end Vcntls_arch;
```

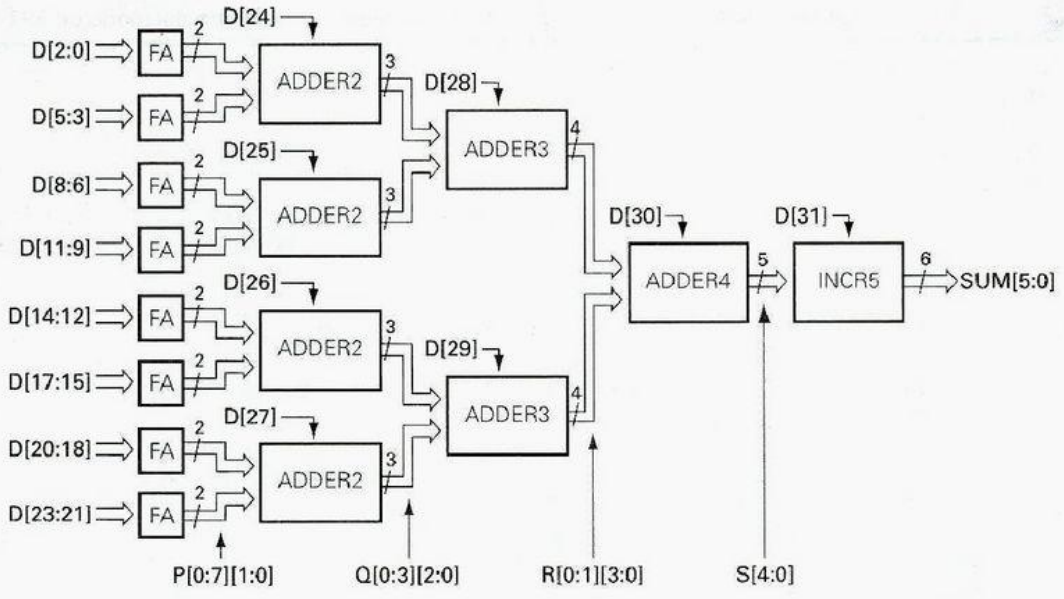


Figura 6-15 Estructura del contador de unos de 32 bits.

Para sintetizar una realización más eficaz del contador de unos, debemos llegar con una estructura eficiente y luego escribir una arquitectura que la describa. Una estructura de tal clase es el árbol sumador ilustrado en la figura 6-15. Un sumador completo (FA, full adder) suma tres bits de entrada para producir una suma de 2 bits. Pares de números de 2 bits se suman mediante sumadores de 2 bits (ADDER2), cada uno de los cuales también tiene una entrada de acarreo que puede incluir sumar otra entrada de 1 bit a su suma. Las sumas resultantes de 3 bits son combinadas mediante sumadores de 3 bits (ADDER3) y el par final de sumas de 4 bits se combinan en un sumador de 4 bits (ADDER4). Al hacer uso de las entradas de acarreo disponibles, esta estructura de árbol puede combinar 31 bits. Un incrementador de 5 bits por separado se emplea al final para manejar el bit de entrada restante.

La estructura de la figura 6-15 puede generarse mediante una arquitectura estructural VHDL, como se muestra en la tabla 6-34. El programa comienza por declarar todos los componentes que se utilizarán en el diseño, los cuales corresponden a los bloques de la figura.

La letra que está debajo de cada columna de señales en la figura 6-15 corresponde al nombre que utiliza esa señal en el programa. Cada una de las señales `P`, `Q`, y `R` es un arreglo con un `STD_LOGIC_VECTOR` por conexión en la columna correspondiente. El programa define un tipo correspondiente para cada una de ellas, seguido por la declaración de señal real.

El programa en la tabla 6-34 hace buen uso de las instrucciones `generate` para crear los múltiples componentes sumador en el lado izquierdo de la figura: ocho FA, cuatro ADDER2 y dos ADDER3. Finalmente, instancia uno de cada ADDER4 e INCR5.

Tabla 6-34
Arquitectura estructural VHDL para un contador de "unos" de 32 bits.

```
architecture Vcntlstr_arch of Vcntlstr is

component FA port ( A, B, CI: in  STD_LOGIC;
                    S, CO:    out STD_LOGIC );
end component;

component ADDER2 port ( A, B: in  STD_LOGIC_VECTOR(1 downto 0);
                        CI:   in  STD_LOGIC;
                        S:    out STD_LOGIC_VECTOR(2 downto 0) );
end component;

component ADDER3 port ( A, B: in  STD_LOGIC_VECTOR(2 downto 0);
                        CI:   in  STD_LOGIC;
                        S:    out STD_LOGIC_VECTOR(3 downto 0) );
end component;

component ADDER4 port ( A, B: in  STD_LOGIC_VECTOR(3 downto 0);
                        CI:   in  STD_LOGIC;
                        S:    out STD_LOGIC_VECTOR(4 downto 0) );
end component;

component INCR5 port ( A:  in  STD_LOGIC_VECTOR(4 downto 0);
                       CI: in  STD_LOGIC;
                       S:  out STD_LOGIC_VECTOR(5 downto 0) );
end component;

type Ptype is array (0 to 7) of STD_LOGIC_VECTOR(1 downto 0);
type Qtype is array (0 to 3) of STD_LOGIC_VECTOR(2 downto 0);
type Rtype is array (0 to 1) of STD_LOGIC_VECTOR(3 downto 0);
signal P: Ptype;  signal Q: Qtype;  signal R: Rtype;
signal S: STD_LOGIC_VECTOR(4 downto 0);

begin
  U1: for i in 0 to 7 generate
    U1C: FA port map (D(3*i), D(3*i+1), D(3*i+2), P(i)(0), P(i)(1));
  end generate;
  U2: for i in 0 to 3 generate
    U2C: ADDER2 port map (P(2*i), P(2*i+1), D(24+i), Q(i));
  end generate;
  U3: for i in 0 to 1 generate
    U3C: ADDER3 port map (Q(2*i), Q(2*i+1), D(28+i), R(i));
  end generate;
  U4: ADDER4 port map (R(0), R(1), D(30), S);
  U5: INCR5 port map (S, D(31), SUM);
end Vcntlstr_arch;
```

Las definiciones de las entidades de componente individual del contador de unos y arquitecturas, desde FA hasta INCR, pueden hacerse en programas estructurales o de comportamiento por separado. Por ejemplo, la tabla 6-35 es un programa estructural para FA. El resto de los componentes quedan como ejercicios (6.20 – 6.22).

Tabla 6-35
Programa VHDL estructural para un sumador completo.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FA is
    port ( A, B, CI: in  STD_LOGIC;
           S, CO:    out STD_LOGIC );
end FA;

architecture FA_arch of FA is
begin
  S <= A xor B xor CI;
  CO <= (A and B) or (A and CI) or (B and CI);
end FA_arch;
```

6.3.7 Juego de "Tres en raya" o "Gato"

Nuestro último ejemplo es el diseño de un circuito combinacional que selecciona el siguiente movimiento de un jugador en el juego de "Tres en raya" o "Gato". En caso de que usted no esté familiarizado con el juego, vea las reglas que se explicaron en el cuadro de la página 492. Repetiremos aquí nuestra estrategia para jugar y ganar el juego:

1. Busque un renglón, columna o diagonal que tenga dos de mis marcas (X o O dependiendo de qué jugador sea yo) y una celda vacía. Si existe una, ponga mi marca en la celda vacía; ¡yo gano!
2. Si no es así, busque un renglón, columna o diagonal que tenga dos de las marcas de mi adversario y una celda vacía. Si existe una, ponga mi marca en la celda vacía para bloquear una victoria potencial de mi oponente.
3. Si no, seleccione una celda basándose en la experiencia. Por ejemplo, si la celda de en medio se encuentra abierta, por lo regular, ocuparla es un buen movimiento. De otro modo, las celdas de las esquinas son buenas opciones. Los jugadores inteligentes también pueden detectar y bloquear un patrón en desarrollo de parte del adversario o "prever" para seleccionar un buen movimiento.

Para evitar la confusión entre "O" y "0" en nuestros programas llamaremos al segundo jugador "Y". Ahora podemos pensar acerca de cómo codificar las entradas y salidas del circuito. Las entradas representan el estado actual del tablero de juego. Existen nueve celdas y cada celda tiene una de tres posibilidades (vacía, ocupada por una X, ocupada por una Y). Las salidas representan el movimiento por hacer, suponiendo que sea el turno de X. Existen solamente nueve posibles movimientos que puede efectuar un jugador, de modo que la salida puede ser codificada con sólo cuatro bits.

Hay varias selecciones de cómo codificar el estado de una celda. Debido a que el juego es simétrico, elegimos una codificación simétrica que puede ayudarnos más adelante:

00 La celda está vacía.
10 La celda se encuentra ocupada con una X.
01 La celda se encuentra ocupada con una Y.

Tabla 6-36
Paquete VHDL con definiciones para el proyecto del tres en raya.

```
library IEEE;
use IEEE.std_logic_1164.all;

package TTTdefs is

type TTTgrid is array (1 to 3) of STD_LOGIC_VECTOR(1 to 3);
subtype TTTmove is STD_LOGIC_VECTOR (3 downto 0);

constant MOVE11: TTTmove := "1000";
constant MOVE12: TTTmove := "0100";
constant MOVE13: TTTmove := "0010";
constant MOVE21: TTTmove := "0001";
constant MOVE22: TTTmove := "1100";
constant MOVE23: TTTmove := "0111";
constant MOVE31: TTTmove := "1011";
constant MOVE32: TTTmove := "1101";
constant MOVE33: TTTmove := "1110";
constant NONE:   TTTmove := "0000";

end TTTdefs;
```

De este modo podemos codificar el estado del tablero de 3 × 3 en 18 bits. Puesto que VHDL soporta arreglos, es útil definir un tipo arreglo, `TTTgrid`, que contiene elementos correspondientes a las celdas en el tablero. Puesto que este tipo será utilizado a lo largo de nuestro proyecto del "tres en raya", es conveniente poner esta definición junto con varias otras que vendrán, en un paquete VHDL, como se muestra en la tabla 6-36.

Sería natural definir `TTTgrid` como un arreglo bidimensional de `STD_LOGIC`, pero no todas las herramientas VHDL respaldan arreglos bidimensionales. En su lugar, lo definimos como un arreglo de `STD_LOGIC_VECTOR`s de 3 bits, los cuales son casi la misma cosa. Para representar el tablero del tres en raya, utilizaremos dos señales `X` y `Y` de este tipo, donde un elemento de una variable es 1, si el jugador así llamado tiene una marca en la celda correspondiente. La figura 6-16 ilustra la correspondencia entre nombres de señal y celdas en el tablero.

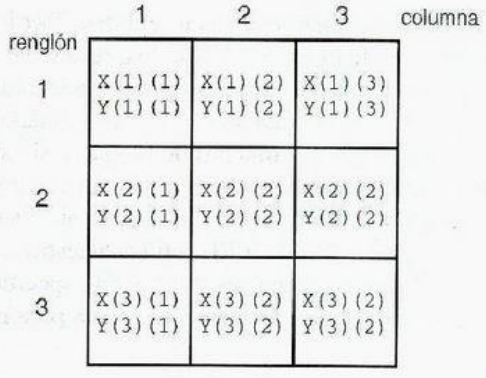


Figura 6-16
Tablero de tres en raya y nombres de señal VHDL.

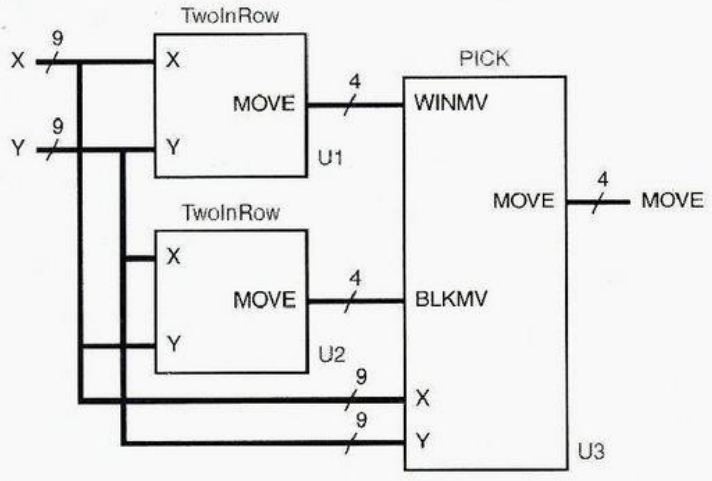


Figura 6-17
Partición de entidad para el juego del "Tres en raya".

El paquete de la tabla 6-36 también define un tipo de 4 bits `TTTmove` para movimientos codificados. Un jugador tiene nueve posibles movimientos, y un código más se emplea para el caso en que ningún movimiento sea posible. Esta codificación particular fue elegida y empleada en el paquete sin otra razón que la misma codificación que fue utilizada en la versión ABEL de este ejemplo en la sección 6.2.7. Al definir primero la codificación en el paquete, después podemos cambiar la definición sin tener que modificar las entidades que la utilizan (por ejemplo, véase el ejercicio 6.23).

Más que tratar de diseñar el circuito para hallar movimientos del tres en raya como una simple entidad monolítica, para nosotros tiene más sentido realizar su partición en segmentos más pequeños. De hecho, dividirlo de acuerdo con los incisos de la estrategia de tres pasos al comienzo de esta sección parece una buena idea.

Cabe indicar que los pasos 1 y 2 de nuestra estrategia son muy similares; difieren solamente en que se invierten los papeles del jugador y el adversario. Una entidad que encuentre un movimiento ganador para mí puede también hallar un movimiento de bloqueo para mi adversario. Observando esta característica desde otro punto de vista, una entidad que encuentre un movimiento ganador para mí puede encontrar un movimiento de bloqueo para mí si las codificaciones para mí y mi oponente se intercambian. Aquí es donde nuestra codificación simétrica vale la pena: podemos intercambiar los jugadores tan sólo con las señales `X` y `Y`.

Con esto en mente, podemos utilizar dos copias de la misma entidad, TwoInRow, para efectuar los pasos 1 y 2 como se ilustra en la figura 6-17. Nótese que la señal X está conectada a la entrada superior de la primera entidad TwoInRow, pero a la entrada inferior de la segunda. Una tercera entidad, `PICK`, selecciona un movimiento ganador, si se tiene disponible uno de U1; si no, selecciona un movimiento de bloqueo, si está disponible de U2; y, si no, utiliza la "experiencia" (paso 3) para seleccionar un movimiento.

La tabla 6-37 es un programa estructural VHDL para la entidad de nivel superior, `GETMOVE`. Además del paquete `std_logic_1164` IEEE, utiliza nuestro paquete `TTTdefs`. Nótese que la cláusula "`use`" para los paquetes `TTTdefs` especifica que se almacena en la biblioteca "`work`", la cual es automáticamente creada para nuestro proyecto.

Tabla 6-37
Entidad VHDL estructural de nivel superior para seleccionar un movimiento en el juego "Tres en raya".

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.TTTdefs.all;

entity GETMOVE is
    port ( X, Y: in  TTTgrid;
           MOVE: out TTTmove );
end GETMOVE;

architecture GETMOVE_arch of GETMOVE is

component TwoInRow port ( X, Y: in  TTTgrid;
                          MOVE: out STD_LOGIC_VECTOR(3 downto 0) );
end component;

component PICK port ( X, Y:          in  TTTgrid;
                      WINMV, BLKMV: in  STD_LOGIC_VECTOR(3 downto 0);
                      MOVE:          out STD_LOGIC_VECTOR(3 downto 0) );
end component;

signal WIN, BLK: STD_LOGIC_VECTOR(3 downto 0);

begin
  U1: TwoInRow port map (X, Y, WIN);
  U2: TwoInRow port map (Y, X, BLK);
  U3: PICK port map (X, Y, WIN, BLK, MOVE);
end GETMOVE_arch;
```

La arquitectura de la tabla 6-37 solamente declara y utiliza dos componentes, `TwoInRow` y `PICK`, la cual definiremos en breve. Las únicas señales internas son `WIN` y `BLK`, que pasan movimientos ganadores y de bloqueo desde las dos instancias de `TwoInRow` a `PICK`, como en la figura 6-17. La parte de instrucciones de la arquitectura tiene apenas tres instrucciones para ejemplificar los tres bloques en la figura.

Ahora viene la parte interesante, el diseño de las entidades individuales en la figura 6-17. Comenzaremos con `TwoInRow`, puesto que explica dos terceras partes del diseño. Su definición de entidad es muy simple, como se muestra en la tabla 6-38. Pero hay mucho que discutir sobre su arquitectura, como se muestra en la tabla 6-39.

Tabla 6-38
Declaración de entidad TwoInRow.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.TTTdefs.all;

entity TwoInRow is
    port ( X, Y: in  TTTgrid;
           MOVE: out TTTmove );
end TwoInRow;
```

Tabla 6-39 Arquitectura de la entidad TwoInRow.

```
architecture TwoInRow_arch of TwoInRow is

function R(X, Y: TTTgrid; i, j: INTEGER) return BOOLEAN is
  variable result: BOOLEAN;
  begin                                -- Encontrar 2 en renglón con celda vacía i, j.
    result := TRUE;
    for jj in 1 to 3 loop
      if jj = j then result := result and X(i)(jj)='0' and Y(i)(jj)='0';
      else result := result and X(i)(jj)='1'; end if;
    end loop;
    return result;
  end R;

function C(X, Y: TTTgrid; i, j: INTEGER) return BOOLEAN is
  variable result: BOOLEAN;
  begin                                -- Encontrar 2 en columna con celda vacía i, j.
    result := TRUE;
    for ii in 1 to 3 loop
      if ii = i then result := result and X(ii)(j)='0' and Y(ii)(j)='0';
      else result := result and X(ii)(j)='1'; end if;
    end loop;
    return result;
  end C;

function D(X, Y: TTTgrid; i, j: INTEGER) return BOOLEAN is
  variable result: BOOLEAN;        -- Encontrar 2 en diagonal con celda vacía i, j.
  begin                            -- Esto es para la diagonal 11, 22, 33.
    result := TRUE;
    for ii in 1 to 3 loop
      if ii = i then result := result and X(ii)(ii)='0' and Y(ii)(ii)='0';
      else result := result and X(ii)(ii)='1'; end if;
    end loop;
    return result;
  end D;

function E(X, Y: TTTgrid; i, j: INTEGER) return BOOLEAN is
  variable result: BOOLEAN;        -- Encontrar 2 en columna con celda vacía i, j.
  begin                            -- Esto es para la diagonal 11, 22, 33.
    result := TRUE;
    for ii in 1 to 3 loop
      if ii = i then result := result and X(ii)(4-ii)='0' and Y(ii)(4-ii)='0';
      else result := result and X(ii)(4-ii)='1'; end if;
    end loop;
    return result;
  end E;
```

Tabla 6-39
(continuación)

```
begin
  process (X, Y)
  variable G11, G12, G13, G21, G22, G23, G31, G32, G33: BOOLEAN;
  begin
    G11 := R(X,Y,1,1) or C(X,Y,1,1) or D(X,Y,1,1);
    G12 := R(X,Y,1,2) or C(X,Y,1,2);
    G13 := R(X,Y,1,3) or C(X,Y,1,3) or E(X,Y,1,3);
    G21 := R(X,Y,2,1) or C(X,Y,2,1);
    G22 := R(X,Y,2,2) or C(X,Y,2,2) or D(X,Y,2,2) or E(X,Y,2,2);
    G23 := R(X,Y,2,3) or C(X,Y,2,3);
    G31 := R(X,Y,3,1) or C(X,Y,3,1) or E(X,Y,3,1);
    G32 := R(X,Y,3,2) or C(X,Y,3,2);
    G33 := R(X,Y,3,3) or C(X,Y,3,3) or D(X,Y,3,3);
    if    G11 then MOVE <= MOVE11;
    elsif G12 then MOVE <= MOVE12;
    elsif G13 then MOVE <= MOVE13;
    elsif G21 then MOVE <= MOVE21;
    elsif G22 then MOVE <= MOVE22;
    elsif G23 then MOVE <= MOVE23;
    elsif G31 then MOVE <= MOVE31;
    elsif G32 then MOVE <= MOVE32;
    elsif G33 then MOVE <= MOVE33;
    else           MOVE <= NONE;
    end if;
  end process;
end TwoInRow_arch;
```

La arquitectura define varias funciones, cada una de las cuales determina si existe un movimiento ganador (desde el punto de vista de X) en una celda particular `i,j`. Existe un movimiento ganador si la celda `i,j` se encuentra vacía y las otras dos celdas en el mismo renglón, columna o diagonal contienen una X. Las funciones `R` y `C` buscan movimientos ganadores en el renglón y columna de la celda `i,j`, respectivamente. Las funciones `D` y `E` buscan en ambas diagonales.

Dentro del único proceso de la arquitectura, se declaran nueve variables `BOOLEAN G11–G33` para indicar si cada una de las celdas tiene un posible movimiento ganador. Las instrucciones de asignación al principio del proceso establecen cada variable a `TRUE` si existe un movimiento así, llamando y combinando todas las funciones apropiadas para la celda `i,j`.

El resto del proceso es una instrucción "`if`" profundamente anidada que busca un movimiento ganador en todas las celdas posibles. Aunque típicamente resulta en lógica sintetizada más lenta, se requiere el "`if`" anidado en lugar de alguna forma de instrucción "case", porque pueden ser posibles múltiples movimientos. Si no es posible un movimiento ganador, se asigna el valor "`NONE`".

IMPUREZA EXPLÍCITA Además de un índice de celda `i,j`, las funciones R, C, D y E en la tabla 6-39 están pasando el estado del tablero X y Y. Esto es necesario porque las funciones son VHDL son por defecto *puras*, lo que significa que las señales y variables declaradas en los padres de la función *no* son directamente visibles dentro de la función. Sin embargo, se puede relajar esta restricción al declarar explícitamente una función como *impura* colocando la palabra clave `impure` antes de la palabra clave `function` en su definición.

La entidad `PICK` combina los resultados de dos entidades `TwoInRow` de acuerdo con el programa en la tabla 6-40. Se da principal prioridad a un movimiento ganador, seguido por un movimiento de bloqueo. De otro modo, la función `MT` se llama para cada celda, comenzando con la de en medio y finalizando con las celdas laterales, para encontrar un movimiento disponible. Esto completa el diseño del circuito del tres en raya.

Tabla 6-40 Programa VHDL para seleccionar un movimiento ganador, de bloqueo o alguno más en el tres en raya usando la "experiencia".

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.TTTdefs.all;

entity PICK is
  port ( X, Y:          in  TTTgrid;
         WINMV, BLKMV: in  STD_LOGIC_VECTOR(3 downto 0);
         MOVE:          out STD_LOGIC_VECTOR(3 downto 0) );
end PICK;

architecture PICK_arch of PICK is
function MT(X, Y: TTTgrid; i, j: INTEGER) return BOOLEAN is
  begin                              -- Determine si la celda i,j se encuentra vacía
    return X(i)(j)='0' and Y(i)(j)='0';
  end MT;
begin
  process (X, Y, WINMV, BLKMV)
  begin                                          -- Si está disponible, seleccione:
    if    WINMV /= NONE then MOVE <= WINMV;      -- movimiento ganador
    elsif BLKMV /= NONE then MOVE <= BLKMV;      -- de otro modo movimiento de bloqueo
    elsif MT(X,Y,2,2)   then MOVE <= MOVE22;     -- de otro modo celda central
    elsif MT(X,Y,1,1)   then MOVE <= MOVE11;     -- de otro modo celdas de esquina
    elsif MT(X,Y,1,3)   then MOVE <= MOVE13;
    elsif MT(X,Y,3,1)   then MOVE <= MOVE31;
    elsif MT(X,Y,3,3)   then MOVE <= MOVE33;
    elsif MT(X,Y,1,2)   then MOVE <= MOVE12;     -- de otro modo celdas laterales
    elsif MT(X,Y,2,1)   then MOVE <= MOVE21;
    elsif MT(X,Y,2,3)   then MOVE <= MOVE23;
    elsif MT(X,Y,3,2)   then MOVE <= MOVE32;
    else                     MOVE <= NONE;       -- de otro modo el tablero
                                                    se encuentra lleno
    end if;
  end process;
end PICK_arch;
```

Ejercicios

6.1 Explique cómo se puede realizar el desplazador rápido de 16 bits de la sección 6. 1. 1 con una combinación de dispositivos 74x157 y 74x151. ¿Cómo se compara este enfoque con los otros en retardo y conteo de partes?

6.2 Muestre cómo se puede realizar el desplazador rápido de 16 bits de la sección 6.1.1 con ocho dispositivos idénticos GAL22V10s.

6.3 Encuentre una codificación de las cantidades de desplazamiento (`S[3:0]`) y modos (`C[2:0]`) en el desplazador rápido en la tabla 6-3 que reduzca adicionalmente el número total de términos de producto utilizados por el diseño.

6.4 Midifique el programa del codificador de prioridad dual de la tabla 6-6 para reducir de forma adicional el número de términos de producto requerido. Establezca si sus cambios incrementan el retardo del circuito cuando se realizan en un dispositivo GAL22V10. ¿Puede reducir los términos de producto de manera suficiente para incluir el diseño dentro en un dispositivo GAL16V8?

6.5 Aquí tenemos un ejercicio donde usted puede usar el cerebro, como lo tuvo que hacer el autor cuando explicó la ecuación para la salida SUM0, en la tabla 6-12. ¿Cada una de las salidas SUM1–SUM3 requiere más o menos términos que SUM0?

6.6 Complete el diseño del circuito contador de unos en ABEL y basado en los PLD, el cual se inició en la sección 6.2.6. Utilice dispositivos PLD como el 22V10 o ciruitos PLD más pequeños para tratar de minimizar el número total de PLD. Calcule el retardo total de su diseño en términos de la cantidad total de retardos (debido a los dispositivos PLD) en el peor de los casos, en una trayectoria de señal que va de entrada a salida.

6.7 Determine otro código para los movimientos del tres en raya (que se muestra en la tabla 6-13) que tengan las mismas propiedades de rotación que el código original. Es decir, debería ser posible compensar para un giro de 180° del tablero utilizando solamente inversores y el reacomodo de alambres. Determine si las ecuaciones `TWOINHAF` todavía se ajustarán en un solo 22V10 haciendo uso del nuevo código.

6.8 Utilizando un simulador, genere una secuencia de movimientos en la cual el PLD PICK2 en la tabla 6-16 perderá un juego de tres en raya, incluso si X juega primero.

6.9 Modifique el programa de la tabla 6-16 para ofrecer al programa una mejor oportunidad de ganar, o por lo menos de no perder. ¿Aún así puede perder el nuevo programa?

6.10 Modifique tanto la "otra lógica" en la figura 6-13 como el programa de la tabla 6-16 para proporcionar al programa una mejor oportunidad de ganar, o por lo menos de no perder. ¿Aún así puede perder su nuevo programa?

6.11 Escriba las funciones VHDL para `Vror`, `Vsll`, `Vsrl`, `Vsla` y `Vsra` en la tabla 6-17 utilizando las operaciones `ror`, `sll`, `srl`, `sla`, y `sra` como se definen en la tabla 6-3.

6.12 La versión del circuito iterativo de un `fixup` en la tabla 6-20 tiene una trayectoria de retardo en el peor de los casos, con 15 compuertas OR, que va desde el primer valor decodificado de `i` (14) hasta la señal `FSEL(0)`. Explique un truco que recorte esta trayectoria de retardo casi a la mitad sin ningún costo adicional (o sin un costo negativo) en las compuertas. ¿Cómo puede extenderse este truco adicionalmente para ahorrar compuertas o entradas de compuertas?

6.13 Vuelva a escribir la definición de la entidad `barrel16` de la tabla 6-17 y la arquitectura de la tabla 6-22 de modo que un bit de control de dirección simple se haga explícitamente disponible para la arquitectura.

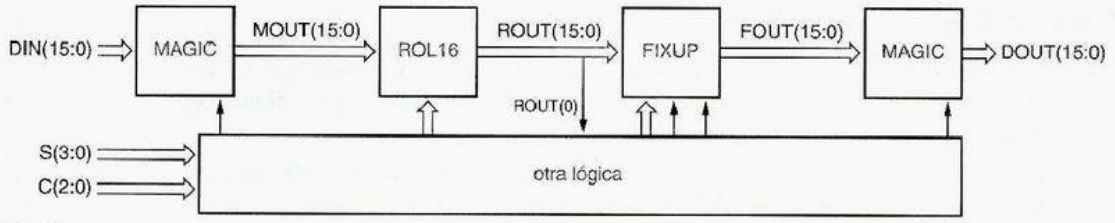


Figura X6.14

6.14 Vuelva a escribir la definición de la arquitectura `barrel16` (ver tabla 6-22) para utilizar la estructura que se indica en la figura X6.14. Utilice las entidades existentes `ROL16` y FIXUP; se deja a su selección encontrar MAGIC y la otra lógica.

6.15 Escriba una versión de semicomportamiento o estructural de la arquitectura `fpencr_arch` de la tabla 6-25 que genere solamente un sumador en síntesis y que no genere comparadores de 10 bits múltiples para la instrucción anidada "`if`".

6.16 Repita el ejercicio 6.15, incluyendo una definición estructural de un circuito de redondeo eficaz que realice la función `round`. Su circuito debe necesitar una cantidad considerablemente menor de compuertas que un sumador de 4 bits.

6.17 Vuelva a diseñar el codificador de prioridad dual VHDL de la sección 6.3.3 para obtener un mejor rendimiento, como se sugiere en el último párrafo de la sección.

6.18 Escriba una arquitectura estructural VHDL para un comparador de 64 bits que sea similar al de la tabla 6-30, con la excepción que ahora el resultado de la comparación se desarrolle en serie desde la etapa menos significativa hasta la más significativa.

6.19 ¿Qué cambio significativo ocurre en la síntesis del programa VHDL en la tabla 6-31 si modificamos las instrucciones en el caso "`when others`" a "`null`"?

6.20 Escriba programas de comportamiento VHDL para los componentes "ADDERx" que se utilizan en la tabla 6-34.

6.21 Escriba programas de comportamiento VHDL para los componentes "ADDERx" de la tabla 6-34. Utilice una definición genérica de modo que la misma entidad pueda ser ejemplificada por ADDER2, ADDER3 y ADDER5, además indique qué cambios deben hacerse en la tabla 6-34 para hacer esto.

6.22 Escriba un programa VHDL estructural para el componente "INCR5" de la tabla 6-34.

6.23 Aplicando una herramienta de síntesis VHDL, sintetice el diseño del tres en raya de la sección 6.3.7, realice lo anterior en un FPGA y determine cuántos recursos internos utiliza. Posteriormente intente reducir los requerimientos de recursos especificando una codificación diferente de los movimientos en el paquete `TTTdefs`.

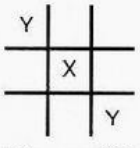
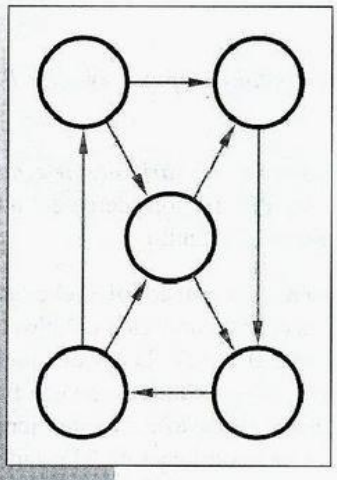


Figura X6.24

6.24 El programa del tres en raya en la sección 6.3.7 finalmente pierde contra un oponente inteligente, si se aplica el estado del tablero que se muestra en la figura X6.24. Utilice un simulador VHDL disponible para demostrar que esto es cierto. Luego modifique la entidad `PICK` para ganar en esta situación y en otras semejantes, luego verifique su diseño empleando el simulador.



capítulo 7

Principios de diseño lógico secuencial

Los circuitos lógicos se clasifican en dos tipos, "combinacional" y "secuencial". Un circuito lógico *combinacional* es aquel cuyas salidas dependen solamente de sus entradas actuales. La perilla giratoria selectora de los canales en un televisor antiguo es como un circuito combinacional: su "salida" selecciona un canal, para esto se basa solamente en su "entrada" actual, la posición de la perilla.

Un circuito lógico *secuencial* es aquel cuyas salidas dependen no sólo de sus entradas actuales, sino también de la secuencia pasada de entradas, arbitrariamente lejos en el pasado. El circuito controlado por los botones de avance o retroceso de canal en un receptor de televisión o en una videocasetera es un circuito secuencial: la selección del canal depende de la secuencia anterior de pulsaciones de avance/retroceso, por lo menos desde que se comenzaron a ver las imágenes con 10 horas de anterioridad, y quizás más hacia atrás hasta el momento en que se conectó el dispositivo en la pared por primera vez.

De modo que es poco práctico, y con frecuencia imposible, describir el comportamiento de un circuito secuencial por medio de una tabla que enumere las salidas como una función de la secuencia de entrada que haya sido recibida hasta el momento presente. Para saber qué hacer a continuación, se necesita saber qué se ha hecho hasta ahora. Con el selector de canales de TV, es imposible determinar qué canal se encuentra seleccionado actualmente observando tan sólo la secuencia anterior de pulsaciones en los botones de avance y retroceso, si observamos las 10 pulsaciones anteriores o las 1,000 anteriores. Se necesita más información, el estado "actual" del selector de canales. Tal vez la mejor definición de "estado" que yo he

visto sea la de Herbert Hellerman, publicada en *Digital Computer Systems Principles* (McGraw-Hill, 1967):

estado
variable de estado

> El *estado* de un circuito secuencial es una colección de *variables de estado* cuyos valores en cualquier tiempo contienen toda la información acerca del pasado necesario para explicar el comportamiento futuro del circuito.

En el ejemplo del selector de canales, el número del canal actual es el estado actual. En los circuitos del TV, este estado se puede almacenar como siete variables de estado binarias que representan un número decimal entre 0 y 127. Dado el estado actual (número de canal), siempre podemos predecir el estado siguiente como una función de las entradas (pulsaciones de los botones de avance/retroceso). En este ejemplo, una salida altamente visible del circuito secuencial es una codificación del estado mismo: el despliegue del número de canal. Otras salidas, internas a la TV, solamente pueden ser funciones combinacionales del estado (por ejemplo, selección de sintonizador de VHF/UHF/cable) o tanto del estado como de la entrada (por ejemplo, apagar la TV si el estado actual es 0 y se pulsa el botón de "retroceso").

Las variables de estado no necesitan significación física directa, y existen con frecuencia muchas maneras de elegirlas para describir un circuito secuencial en particular. Por ejemplo, en el selector de canales de la TV, el estado puede almacenarse como tres dígitos BCD o 12 bits, con muchas de las combinaciones de bit (4,096 posibles) sin utilizar.

En un circuito de lógica digital, las variables de estado son valores binarios, correspondientes a ciertas señales lógicas en el circuito, como veremos en secciones posteriores. Un circuito con n variables de estado binarias tiene 2^n estados posibles. Tan grande como parezca, 2^n siempre será finito, nunca infinito, de modo que los circuitos secuenciales en ocasiones se conocen como *máquinas de estado finito*.

máquina de estado finito

reloj

Los cambios de estado de la mayoría de los circuitos secuenciales se presentan en tiempos especificados por una señal de *reloj* de funcionamiento libre. La figura 7-1 nos presenta los diagramas de temporización y nomenclatura para señales de reloj típicas. Por convención, una señal de reloj es de estado activo alto si los cambios de estado se presentan en el flanco de subida del reloj o cuando el reloj está en ALTO, y de estado activo bajo en el caso complementario. El *periodo del reloj* es el tiempo entre transiciones sucesivas en la misma dirección, y la *frecuencia del reloj* es el recíproco del periodo. El primer flanco o pulso en un periodo de reloj o en ocasiones el periodo mismo se denomina una *marca ("tic") de reloj*. El *lazo de trabajo* es el porcentaje de tiempo que la señal de reloj se encuentra en su nivel asertivo. Los sistemas digitales típicos, desde relojes digitales hasta supercomputadoras, utilizan un oscilador de cristal de cuar-

periodo de reloj
frecuencia de reloj

marca ("tic") de reloj
lazo de trabajo

MÁQUINAS DE ESTADO NO FINITAS

Un grupo de matemáticos propuso recientemente una máquina de estado no finita, pero todavía se encuentran ocupados enumerando sus estados.... Perdón, era sólo una broma. *Existen* modelos matemáticos para máquinas de estado infinito, como las máquinas de Turing. Por lo regular contienen una pequeña unidad de control de máquina de estado finito, y una cantidad infinita de memoria auxiliar, como una cinta sin fin.

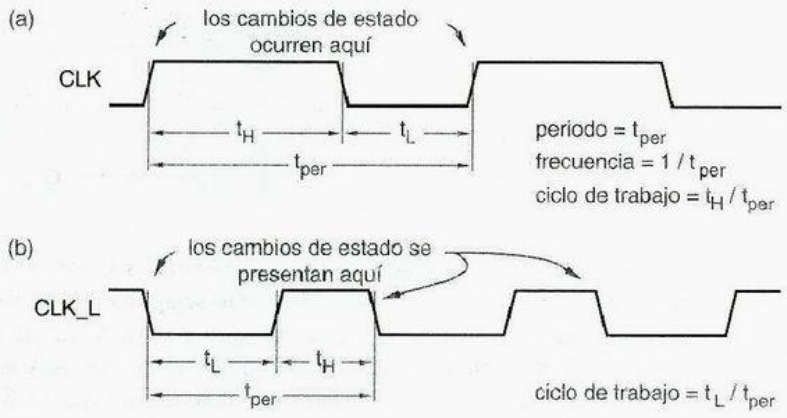


Figura 7-1
Señales de reloj:
(a) de nivel activo alto;
(b) de nivel activo bajo.

zo para generar una señal de reloj de funcionamiento libre. Las frecuencias de reloj pueden abarcar desde 32.768 kHz (para un reloj de pulsera, hasta 500 MHz (para un microprocesador RISC CMOS con un tiempo de ciclo de 2 ns); los sistemas típicos que emplean partes TTL y CMOS tienen frecuencias de reloj en el intervalo de los 5-150 MHz.

En este capítulo discutiremos dos tipos de circuitos secuenciales que engloban a la mayoría de los diseños discretos prácticos. Un *circuito secuencial de retroalimentación* utiliza compuertas ordinarias y lazos de retroalimentación para obtener memoria en un circuito lógico, con lo que crea bloques de construcción de circuito secuencial, como los cerrojos y flip-flops, que son empleados en diseños de más alto nivel. Una *máquina de estado sincrónica temporizada* utiliza estos bloques de construcción en particular los flip-flops D disparados por flanco, para crear circuitos cuyas entradas son examinadas y cuyas salidas cambian de acuerdo con una señal de reloj de control. Otros tipos de circuito secuencial como los del modo fundamental general, modo de pulso múltiple y circuitos multifásicos, son, en ocasiones, útiles en sistemas de alto rendimiento y VLSI; se estudian en textos avanzados.

circuito secuencial de retroalimentación

máquina de estado sincrónica temporizada

7.1 Elementos biestables

El circuito secuencial más sencillo consiste en un par de inversores que forman un lazo de retroalimentación, como se muestra en la figura 7-2. *No* tiene entradas y tiene dos salidas, Q y Q_L.

7.1.1 Análisis digital

El circuito de la figura 7-2 se denomina con frecuencia un *biestable*, puesto que un análisis estrictamente digital muestra que tiene dos estados estables. Si Q es ALTO, entonces el inversor inferior tiene una entrada en ALTO y una salida en BAJO, lo que fuerza al nivel ALTO la salida del inversor superior, como supusimos en primer lugar. Pero si Q está en BAJO, entonces el inversor inferior tiene una entrada en BAJO y una salida en ALTO, lo que fuerza Q a BAJO, otra situación estable. Podríamos utilizar una variable de estado simple, el estado de la señal Q, para describir el estado del circuito; existen dos estados posibles, Q = 0 y Q = 1.

biestable

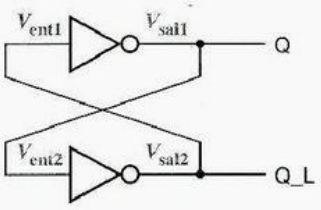


Figura 7-2
Un par de inversores formando un elemento biestable.

El elemento biestable es tan simple que no tiene entradas y por consiguiente no hay manera de controlar o modificar su estado. Cuando se aplica inicialmente la energía al circuito, llega aleatoriamente a un estado o al otro y se mantiene ahí permanentemente. Con todo, sirve muy bien para nuestros propósitos ilustrativos, y *mostraremos* en realidad un par de aplicaciones para el mismo en las secciones 8.2.3 y 8.2.4.

7.1.2 Análisis analógico

El análisis del biestable tiene más que mostrar si consideramos su funcionamiento desde un punto de vista analógico. La línea oscura en la figura 7-3 muestra la función de transferencia T de estado estacionario (DC) para un inversor simple; el voltaje de salida es una función del voltaje de entrada, $V_{sal} = T(V_{ent})$. Con dos inversores conectados en un lazo de retroalimentación como en la figura 7-2, sabemos que $V_{ent1} = V_{sal2}$ = y V_{ent2} V_{sal1}; por lo tanto, podemos graficar las funciones de transferencia para ambos inversores en la misma gráfica con una apropiada rotulación de los ejes. De este modo, la línea oscura es la función de transferencia para el inversor superior en la figura 7-2, y la línea de color gris es la función de transferencia para el inferior.

Considerando solamente el comportamiento de estado estacionario de lazo de retroalimentación del biestable, y no los efectos dinámicos, el lazo se encuentra en equilibrio si los voltajes de entrada y salida de ambos inversores son valores constantes de DC consistentes con la conexión de lazo y la función de transferencia DC de los inversores. Es decir, debemos tener

$$\begin{aligned} V_{ent1} &= V_{sal2} \\ &= T(V_{ent2}) \\ &= T(V_{sal1}) \\ &= T(T(V_{ent1})) \end{aligned}$$

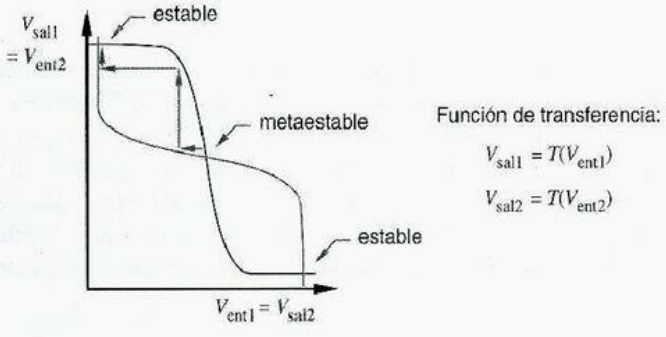


Figura 7-3
Funciones de transferencia para inversores en un lazo de retroalimentación biestable.

Igualmente, debemos tener

$$V_{ent2} = T(T(V_{ent2}))$$

Podemos hallar estos puntos de equilibrio gráficamente de la figura 7-3; que son los puntos en los cuales se cruzan las dos curvas de transferencia. Sorprendentemente, encontramos que no solamente hay dos, sino *tres* puntos de equilibrio. Dos de éstos, denominados *estables*, corresponden a los dos estados que nuestro análisis "estrictamente digital" identificó con anterioridad, con Q siendo 0 (BAJO) o 1 (ALTO).

estable

El tercer punto de equilibrio, denominado *metaestable*, se presenta con V_{sal1} y V_{sal2} aproximadamente la mitad del camino entre un voltaje 1 lógico válido y un voltaje de 0 lógico; de manera que Q y Q_L no son señales lógicas válidas en este punto. Todavía se satisfacen las ecuaciones de ciclo; si podemos conseguir que el circuito funcione en el punto metaestable, en teoría podría permanecer allí indefinidamente.

metaestable

7.1.3 Comportamiento metaestable

Una análisis más detallado de la situación en el punto metaestable muestra que está acertadamente nombrada. No es verdaderamente estable, porque el ruido aleatorio tendrá tendencia a llevar un circuito que se encuentre funcionando en el punto metaestable hacia uno de los puntos de operación estables, como lo demostraremos a continuación.

Supongamos que el biestable se encuentra funcionando precisamente en el punto metaestable en la figura 7-3. Ahora supongamos que una pequeña cantidad del ruido del circuito reduce V_{ent1} por una cantidad diminuta. Este diminuto cambio provoca que V_{sal1} se *incremente* por una cantidad pequeña. Pero puesto que V_{sal1} produce V_{ent2}, podemos seguir la primera flecha horizontal cercana al punto metaestable para la segunda característica de transferencia, lo que ahora demanda un voltaje más bajo para V_{sal2}, el cual es V_{ent1}. Ahora nos encontramos de regreso en donde comenzamos, excepto que tenemos un cambio mucho mayor en voltaje para V_{ent1} que el ruido original producido, y el punto de operación se encuentra cambiando todavía. Este proceso "regenerativo" continúa hasta que alcanzamos el punto de operación estable en la esquina superior izquierda de la figura 7-3. Sin embargo, si realizamos un análisis de "ruido" para cualquiera de los puntos de operación estable, encontramos que la retroalimentación devuelve el circuito hacia el punto de operación estable, en vez de alejarlo de allí.

El comportamiento metaestable de un biestable puede compararse con el comportamiento de una pelota que se deja caer sobre una colina, como se muestra en la figura 7-4. Si dejamos caer la pelota desde la cima, probablemente rodará de inmediato hacia abajo hacia uno u otro lado de la elevación. Pero si llega a alcanzar justo la cima, puede mantenerse precariamente ahí por un momento antes de que de manera aleatoria las fuerzas

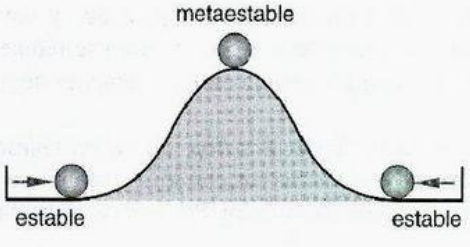


Figura 7-4
Analogía de pelota y colina para ilustrar el comportamiento metaestable.

(del aire, de roedores, terremotos) comiencen a desplazarla hacia la parte inferior de la colina. Como la pelota en la cima de esta colina, el biestable puede permanecer en el estado metaestable durante un periodo impredecible antes que se establezca de manera no determinista en un estado estable o en el otro.

Si el circuito secuencial *más sencillo* es susceptible de un comportamiento metaestable, se puede estar seguro de que *todos* los circuitos secuenciales son susceptibles. Y este comportamiento no es algo que solamente ocurra en el encendido.

Regresando a la analogía de la pelota y la colina, considere qué pasaría si intentamos patear la pelota de un lado al otro de la colina. Aplique una gran fuerza (tipo Arnold Schwarzenegger) y la pelota pasará sobre la cima y aterrizará en un sitio de reposo estable en el otro lado. Aplique una fuerza débil (tipo Mr. Bean) y la pelota caerá de vuelta en el sitio original de arranque. Pero si se aplica una fuerza regular, floja (tipo Charlie Brown) la pelota llegará a la cima de la colina, vacilará, y finalmente descenderá hacia un lado o hacia el otro.

Este comportamiento es completamente análogo a lo que ocurre en los flip-flops bajo condiciones de disparo mínimas. Por ejemplo, pronto estudiaremos los flip-flops S-R, donde un pulso en la entrada S obliga al flip-flop a ir del estado 0 al estado 1. Una anchura de pulso mínimo es especificada para la entrada S. Aplique un pulso de esta anchura o mayor y el flip-flop irá de inmediato al estado 1. Aplique un pulso muy corto, el flip-flop permanecerá en el estado 0. Si se aplicara un pulso apenas por debajo de la anchura mínima, el flip-flop llegaría hasta el estado metaestable. Una vez que el flip-flop se encuentra en el estado metaestable, su funcionamiento depende de "la forma de la colina". Los flip-flops construidos a partir de tecnologías rápidas de alta ganancia, tienden a salir de la metaestabilidad con más rapidez que aquellos construidos a partir de tecnologías con menor rendimiento.

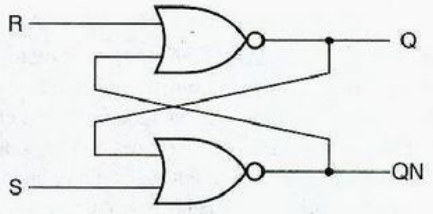
Diremos más acerca de la metaestabilidad en la siguiente sección en relación con tipos específicos de flip-flops, y en la sección 8.9 con respecto a la metodología de diseño sincrónico y fallas de sincronizador.

7.2 Latchs y flip-flops

Los latchs y flip-flops son los bloques de construcción básicos de la mayoría de los circuitos secuenciales. Los sistemas digitales típicos hacen uso de latchs y flip-flops que son dispositivos preencapsulados, especificados funcionalmente en un circuito integrado estándar. En ambientes de diseño ASIC, los latchs y flip-flops son por lo regular celdas predefinidas especificadas por el vendedor de ASIC. Sin embargo, dentro de un CI o ASIC estándar, cada *latch* o flip-flop se diseña típicamente como si fuera un circuito secuencial de retroalimentación que utiliza compuertas lógicas individuales y lazos de retroalimentación. Estudiaremos estos diseños discretos por dos razones: para comprender mejor el comportamiento de los elementos preencapsulados y para obtener la capacidad de construir un latch o flip-flop "desde cero", como se requiere en ocasiones en la práctica del diseño digital y con frecuencia en los exámenes de diseño digital.

flip-flop (biestable)

Todos los diseñadores digitales utilizan el nombre *flip-flop* (o *elemento biestable*) para un dispositivo secuencial que normalmente muestrea sus entradas y cambia sus salidas solamente en ocasiones determinadas por una señal de reloj. Por otra parte, la mayoría



S	R	Q	QN
0	0	última Q	última QN
0	1	0	1
1	0	1	0
1	1	0	0

(b)

Figura 7-5
Latch S-R: a) diseño de circuito que utiliza compuertas NOR; b) tabla de función.

de los diseñadores digitales utilizan el nombre *latch* para un dispositivo secuencial que monitorea todas sus entradas continuamente y modifica sus salidas en cualquier momento, de manera independiente de una señal de reloj. Seguiremos esta convención estándar en este texto. Sin embargo, algunos textos y diseñadores digitales pueden emplear (incorrectamente) el nombre "flip-flop" para un dispositivo que denominamos "latch".

latch

En cualquier caso, debido a que los comportamientos funcionales de los latchs y flip-flops son bastante diferentes, es importante para el diseñador lógico saber cuál tipo está siendo empleado en un diseño, ya sea por medio del número de parte del dispositivo (por ejemplo, 74x374 vs 74x373) o de otra información contextual. Discutiremos los tipos más comúnmente utilizados de latchs y flip-flops en las subsecciones siguientes.

7.2.1 Latch S-R

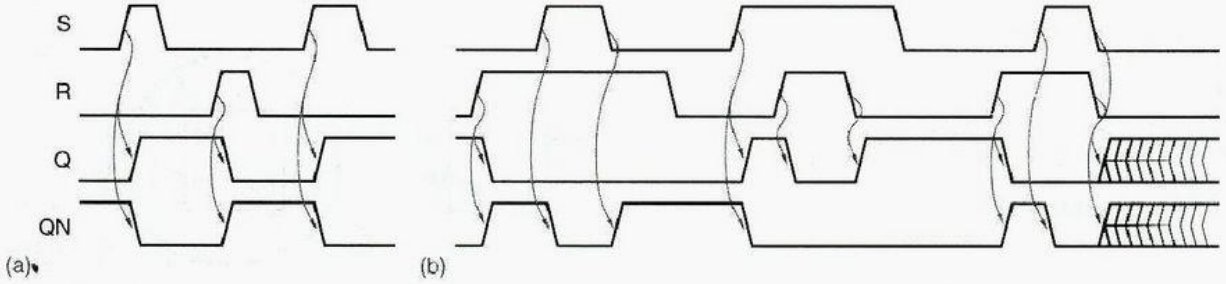
Un latch *S-R (set-reset, de establecimiento-restablecimiento)* basado en compuertas NOR se ilustra en la figura 7-5(a). El circuito tiene dos entradas, S y R, y dos salidas, etiquetadas como Q y QN, donde QN es normalmente el complemento de Q. La señal QN se representa en ocasiones como $\overline{Q}$ o Q_L.

latch S-R

Si tanto S como R son 0, el circuito se comporta como el elemento biestable: tendremos un lazo de retroalimentación que retiene uno o dos estados lógicos, Q = 0 o Q = 1. Como se muestra en la figura 7-5(b), tanto S como R pueden ser asertivas para forzar al lazo de retroalimentación a un estado deseado. S *establece* o *prestablece* la salida Q a 1; R *restablece* o *limpia* la salida Q a 0. Después de que la entrada S o R es negada, el latch permanece en el estado al cual fue forzado. La figura 7-6(a) ilustra el comportamiento funcional de un latch S-R para una secuencia típica de entradas. Las flechas color gris indican causalidad, es decir, cuáles transiciones de entrada provocan ciertas transiciones de salida.

establecer
prestablecer
restablecer
limpiar o borrar

Figura 7-6 Funcionamiento típico de un latch S-R: a) entradas "normales"; b) S y R afirmadas simultáneamente.



$\overline{Q}$ VERSUS QN En la mayoría de las aplicaciones de un latch S-R, la salida QN (alias $\overline{Q}$) es siempre el complemento de la salida Q. Sin embargo, el nombre $\overline{Q}$ no es tan correcto, porque hay un caso donde esta salida no es el complemento de Q. Si tanto S como R son 1, como lo son en varios sitios en la figura 7-6(b), entonces ambas salidas son forzadas a 0. Una vez que negamos cualquiera de las entradas, las salidas regresan a la operación complementaria como es habitual. Sin embargo, si negamos ambas entradas de manera simultánea, el latch se encamina a un estado siguiente impredecible, y de hecho puede oscilar o entrar al estado metaestable. La metaestabilidad también puede presentarse si un pulso 1 que sea demasiado breve se aplica a S o R.

Se muestran tres diferentes símbolos lógicos para el mismo circuito de latch S-R en la figura 7-7. Los símbolos difieren en el tratamiento de la salida complementada. Históricamente, se utilizaba el primer símbolo, mostrando el nombre de señal complementada a nivel activo bajo dentro del rectángulo de función. Sin embargo, en el diseño lógico en burbuja se prefiere la segunda forma del símbolo, mostrando una burbuja de inversión en el exterior del rectángulo de función. La última forma del símbolo es obviamente errónea.

retardo de propagación

La figura 7-8 define los parámetros de temporización para un latch S-R. El *retardo de propagación* es el tiempo que le toma a una transición en una señal de entrada producir una transición en una señal de salida. Un latch o flip-flop dados pueden tener varias especificaciones de retardo de propagación diferentes, una para cada par de señales de entrada y salida. También, el retardo de propagación puede ser diferente dependiendo de si la salida implica una transición BAJO a ALTO o ALTO a BAJO. Con un latch S-R, una transición BAJO a ALTO en S puede ocasionar una transición BAJO a ALTO en Q, de modo que se presenta un retardo de propagación $t_{pLH(SQ)}$, como se muestra en la transición 1 de la figura. De manera semejante, una transición BAJO a ALTO en R puede provocar una transición ALTO a BAJO en Q, con un retardo de propagación $t_{pHL(RQ)}$ como se muestra en la transición 2. En la figura no se muestran las transiciones correspondientes a QN, que tendría retardos de propagación $t_{pHL(SQN)}$ y $t_{pHL(RQN)}$.

anchura de pulso mínima

Las especificaciones de anchura de pulso mínima se proporcionan generalmente para las entradas S y R. Como se puede observar en la figura 7-8, el latch puede ir hacia el estado metaestable y permanecer allí por un periodo aleatorio si se aplica a S o R un pulso más breve que la anchura mínima $t_{pw(min)}$. El latch puede sacarse solamente del estado metaestable mediante la aplicación de un pulso a S o R que iguale o exceda el requerimiento de anchura de pulso mínima.

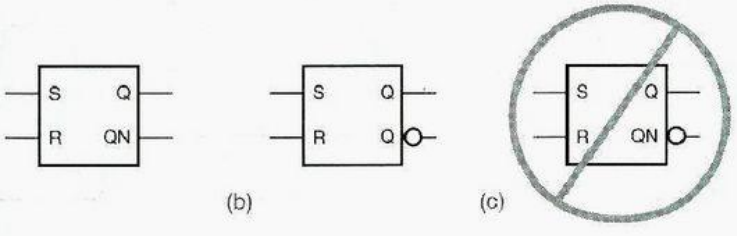


Figura 7-7
Símbolos para un latch S-R: a) sin burbuja; b) preferida para el diseño en burbuja; c) incorrecto debido a la doble negación.

¿QUÉ TAN CERCA ES CERCA?

Como se mencionó en la nota anterior, un latch S-R puede entrar al estado metaestable si S y R se niegan de manera simultánea. Con frecuencia, pero no siempre, las especificaciones de un latch comercial definen "simultáneamente" (por ejemplo, S y R negadas dentro de 20 ns de diferencia entre ellas). Este parámetro en ocasiones se denomina el *tiempo de recuperación*, t_{rec}. Es el mínimo retardo entre la negación de S y R para que sean consideradas no simultáneas y está estrechamente relacionado con la especificación de anchura de pulso mínima. Ambas especificaciones son medidas de cuánto tiempo le toma el ciclo de retroalimentación del latch estabilizarse durante un cambio de estado.

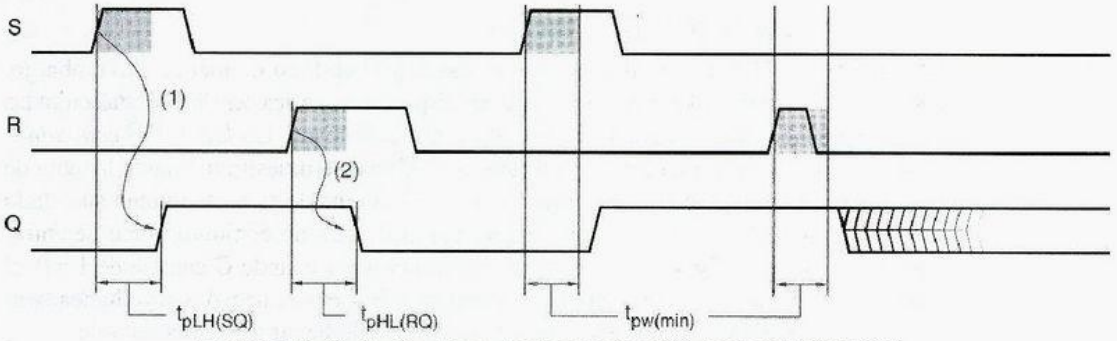


Figura 7-8 Parámetros de temporización para un latch S-R.

7.2.2 Latch S-R

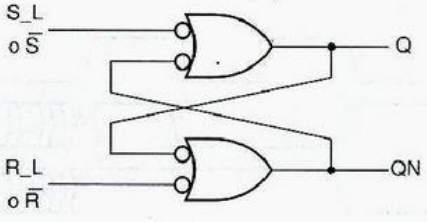
latch $\overline{S}$-$\overline{R}$

Un *latch* $\overline{S}$-$\overline{R}$ (léase "latch S-barra R-barra") con entradas de establecimiento y restablecimiento de nivel activo bajo puede construirse a partir de compuertas NAND como se ilustra en la figura 7-9(a). En las familias lógicas TTL y CMOS, los latchs $\overline{S}$-$\overline{R}$ son utilizados con mucha más frecuencia que los latchs S-R porque las compuertas NAND son preferidas sobre las compuertas NOR.

Como se muestra en la tabla de función, en la figura 7-9(b), el funcionamiento del latch $\overline{S}$-$\overline{R}$ es semejante al del S-R, con dos diferencias principales. La primera, $\overline{S}$ y $\overline{R}$ son de nivel activo bajo, de modo que el latch recuerda su estado anterior cuando $\overline{S} = \overline{R} = 1$; las entradas de nivel activo bajo se indican claramente en los símbolos del inciso c). En segundo lugar, cuando tanto $\overline{S}$ como $\overline{R}$ son afirmadas de manera simultánea, ambas salidas del latch se van a 1 no a 0 como en el latch $\overline{S}$-$\overline{R}$. Exceptuando estas diferencias, el funcionamiento del S-R es el mismo del S-R, incluyendo las consideraciones de temporización y metaestabilidad.

Figura 7-9 Latch $\overline{S}$-$\overline{R}$: a) circuito diseñado con las compuertas NAND; b) tabla de función; c) símbolo lógico.

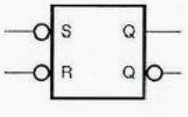
(a)

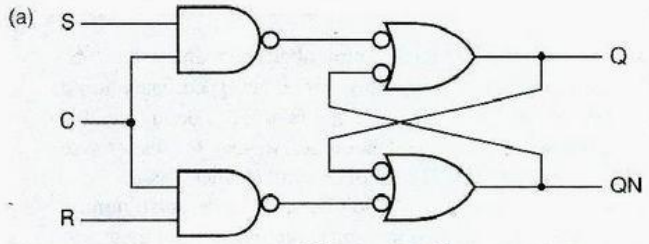


(b)

S_L	R_L	Q	QN
0	0	1	1
0	1	1	0
1	0	0	1
1	1	última Q	última QN

(c)





(b)

S	R	C	Q	QN
0	0	1	última Q	última QN
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
x	x	0	última Q	última QN

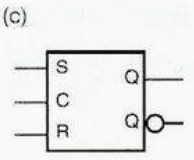


Figura 7-10 Latch S-R con habilitación: a) circuito que utiliza compuertas NAND; b) tabla de función; c) símbolo lógico.

7.2.3 Latch S-R con habilitación

latch S-R con habilitación

Un latch $\overline{S}$-$\overline{R}$ o S-R es sensible a sus entradas S y R en todo momento. Sin embargo, puede ser fácilmente modificado para crear un dispositivo que sea sensible a estas entradas solamente cuando una entrada C de habilitación sea afirmada. Un *latch S-R con habilitación* de esta clase se muestra en la figura 7-10. Como se muestra mediante la tabla de función, el circuito se comporta como un latch S-R cuando C es 1, retiene su estado anterior cuando C es 0. El comportamiento del latch para un conjunto típico de entradas se muestra en la figura 7-11. Si tanto S como R son 1 cuando C cambia de 1 a 0, el circuito se comporta como un latch S-R en el cual S y R son negadas simultáneamente: el estado siguiente es impredecible y la salida puede llegar a ser metaestable.

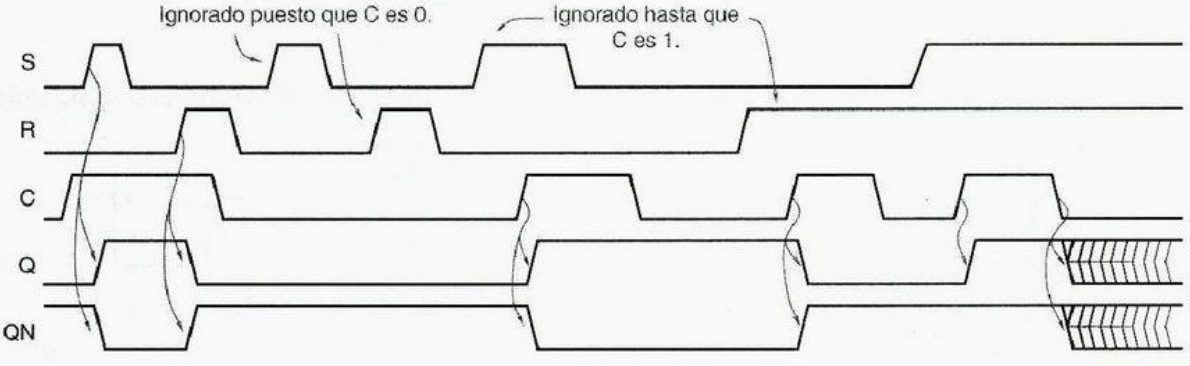
7.2.4 Latch D

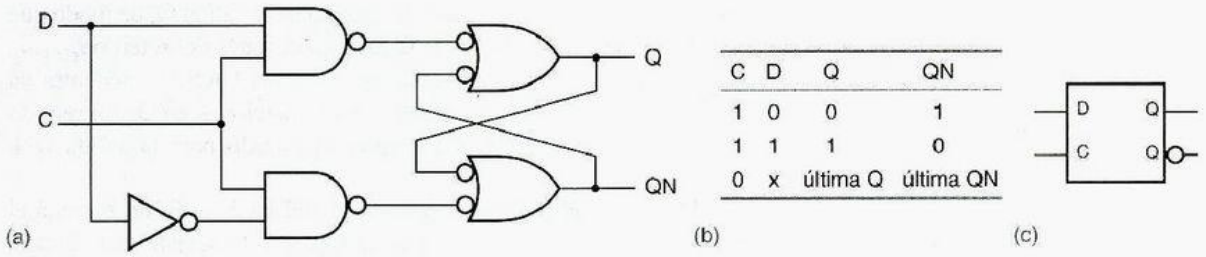
Los latchs S-R son útiles en aplicaciones de control, donde con frecuencia pensamos en términos de establecer un indicador o "bandera" en respuesta a alguna condición y restablecerla cuando cambian las condiciones. De este modo, controlamos las entradas de establecimiento y restablecimiento de manera algo independiente. Sin embargo, con frecuencia necesitamos latchs simplemente para almacenar bits de información: cada bit es presentado en una línea de señal, y nos gustaría almacenarlo en alguna parte. Un *latch D* puede emplearse en una aplicación de esta naturaleza.

latch D

La figura 7-12 muestra un latch D. Su diagrama lógico se reconoce como el de un latch S-R con habilitación, con un inversor agregado para generar entradas S y R a

Figura 7-11 Funcionamiento típico de un latch S-R con habilitación.





C	D	Q	QN
1	0	0	1
1	1	1	0
0	x	última Q	última QN

Figura 7-12 Latch D: a) diseño de círculo empleando compuertas, NAND; b) tabla de función, c) símbolo lógico.

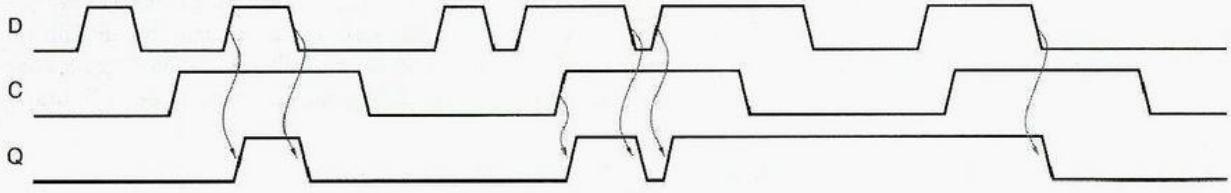


Figura 7-13 Comportamiento funcional de un latch D para varias entradas.

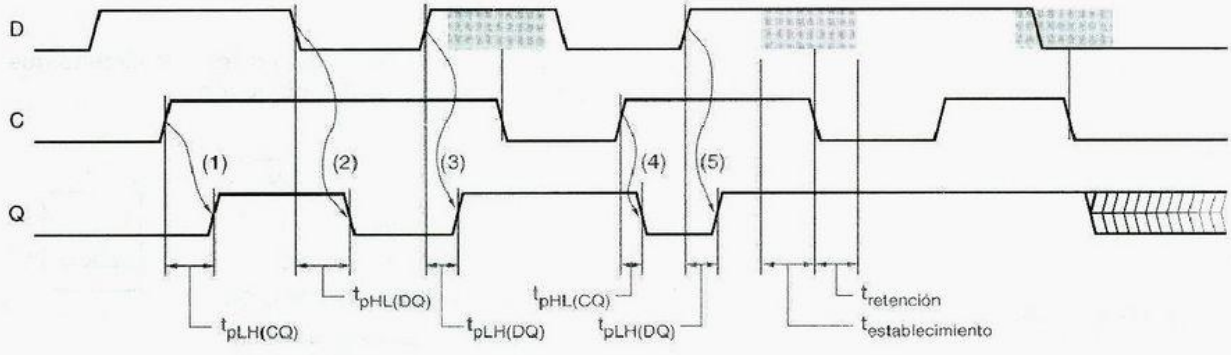
partir de la entrada simple (de datos) D. Esto elimina la situación problemática en los latchs S-R, donde S y R pueden afirmarse de manera simultánea. La entrada de control de un latch D, etiquetada C en c), se llama en ocasiones ENABLE, CLK o G y es de nivel activo bajo en algunos diseños de latch D.

Un ejemplo del comportamiento funcional del latch D se ofrece en la figura 7-13. Cuando la entrada C es afirmada, la salida Q sigue la entrada D. En esta situación, se dice que el latch está "abierto" y la trayectoria desde la entrada D hasta la salida Q es "transparente"; el circuito se denomina con frecuencia un *latch transparente* por esta razón. Cuando la entrada C es negada, el latch se "cierra"; la salida Q retiene su último valor y ya no cambia en respuesta a D, mientras que C permanece negada.

latch transparente

Un comportamiento de temporización más detallado se muestra en la figura 7-14. Cuatro diferentes parámetros de retardo se ilustran para señales que se propagan desde la entrada C o D hasta la salida Q. Por ejemplo, para las transiciones 1 y 4 el latch se

Figura 7-14 Parámetros de temporización para un latch D.



encuentra inicialmente "cerrado" y la entrada D es la opuesta de la salida Q, de modo que cuando C tiende a 1 el latch "se abre" y la salida Q cambia después del retardo $t_{pLH(CQ)}$ o $t_{pHL(CQ)}$. Para las transiciones 2 y 3 la entrada C ya es 1 y el latch se encuentra ya abierto, de modo que Q sigue de manera transparente las transiciones en D con retardo $t_{pHL(DQ)}$ y $t_{pLH(DQ)}$. Cuatro parámetros más especifican el retardo para la salida QN, que no se muestran.

Aunque el latch D elimina el problema S = R = 1 del latch S-R, no elimina el problema de la metaestabilidad. Como se ilustra en la figura 7-14, existe una ventana (sombreada) de tiempo alrededor del borde o flanco descendente de C cuando la entrada D no debe cambiar. Esta ventana comienza en el tiempo $t_{establecimiento}$ antes del flanco descendente (de cierre) de C; $t_{establecimiento}$ se denomina el *tiempo de establecimiento*. La ventana termina en el tiempo $t_{retención}$ posteriormente; $t_{retención}$ se conoce como el *tiempo de retención*. Si D cambia en cualquier momento durante la ventana del tiempo de establecimiento y retención, la salida del latch será impredecible y puede llegar a convertirse en metaestable, como se muestra para el último flanco de cierre de la figura.

tiempo de establecimiento

tiempo de retención

7.2.5 Flip-flop D disparado por flanco

flip-flop D disparado por flanco positivo

Un *flip-flop D disparado por flanco positivo* combina un par de latchs D, como se ilustra en la figura 7-15, para crear un circuito que muestrea su entrada D y cambia sus salidas Q y QN sólo para el flanco ascendente de una señal CLK de control. El primer latch se conoce como el *maestro*; se abre y sigue la entrada cuando CLK es 0. Cuando CLK sube a 1, el latch maestro se cierra y su salida se transfiere al segundo latch conocido como el *esclavo*. El latch esclavo se encuentra abierto todo el tiempo que CLK es 1, pero solamente cambia al principio de este intervalo, porque el maestro está cerrado y sin modificaciones durante el resto del intervalo.

maestro

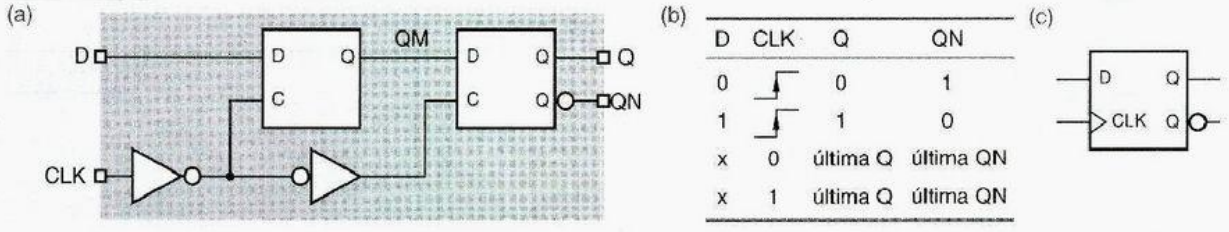
esclavo

El triángulo en la entrada CLK de los flip-flops D indica un comportamiento de disparo por flanco y se denomina *indicador de entrada dinámica*. Ejemplos del comportamiento funcional del flip-flop para diversas transiciones de entrada se presentan en la figura 7-16. La señal QM es la salida del latch maestro. Nótese que QM cambia solamente cuando CLK es 0. Cuando CLK se va a 1, el valor actual de QM se transfiere a Q, y se evita que QM cambie hasta que CLK regresa de nuevo al valor de 0.

indicador de entrada dinámica

La figura 7-17 muestra un comportamiento de temporización más detallado para el flip-flop D. Todos los retardos de propagación se miden desde el flanco ascendente de CLK, puesto que ése es el único evento que ocasiona un cambio de salida. Los diferentes retardos pueden especificarse para cambios de salida BAJO a ALTO y ALTO a BAJO.

Figura 7-15 Flip-flop D disparado por flanco positivo: a) diseño de circuito que utiliza latchs D; b) tabla de función; c) símbolo lógico.



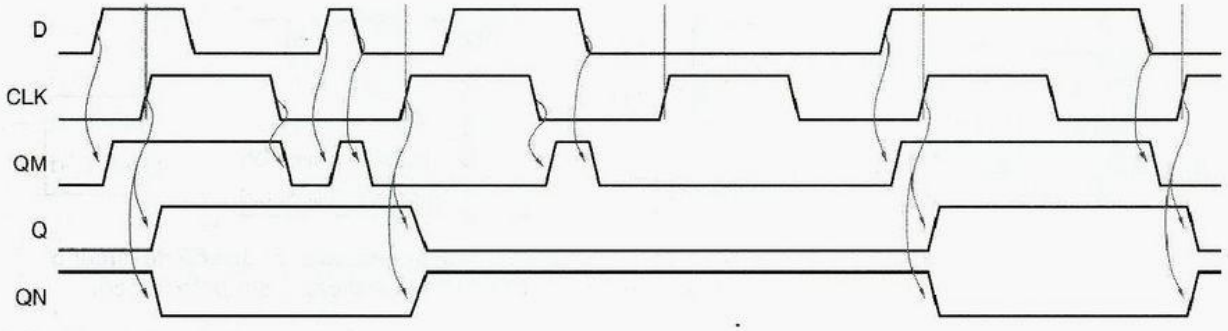


Figura 7-16 Comportamiento funcional de un flip-flop D disparado por flanco positivo.

Como un latch D, el flip-flop D disparado por flanco tiene una ventana de tiempos de establecimiento y de retención durante los cuales las entradas D no deben cambiar. Esta ventana se presenta en torno del flanco de disparo de CLK, y se indica mediante un sombreado de color gris en la figura 7-17. Si los tiempos de establecimiento y retención no se satisfacen, la salida del flip-flop por lo regular se irá a un estado estable, aunque impredecible, de 0 a 1. Sin embargo, en algunos casos la salida oscilará o se irá a un estado metaestable a mitad del camino entre 0 y 1, como se muestra del segundo al último ciclo de reloj en la figura. Si el flip-flop tiende al estado metaestable, regresará a un estado estable por sí mismo después de un retardo probabilístico, como se explica en la sección 8.9. También puede forzarse a un estado estable mediante la aplicación de otro flanco de reloj disparado con una entrada D que satisfaga los requerimientos del tiempo de establecimiento y de retención, como se ilustra en la figura para el último ciclo de reloj.

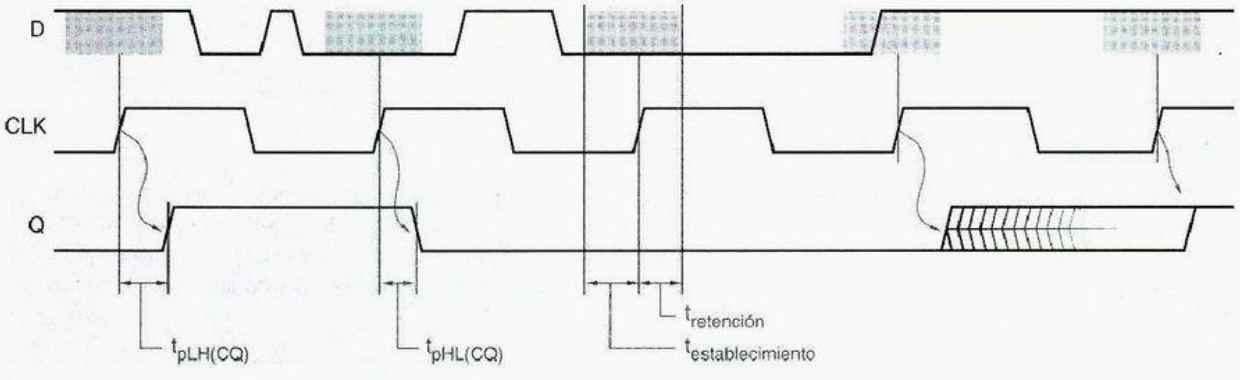
flip-flop D disparado por flanco negativo

Un *flip-flop D disparado por flanco negativo* simplemente invierte la entrada de reloj, de modo que toda la acción se desarrolla sobre el flanco descendente de CLK_L; por convención, un disparador de flanco descendente se considera como de nivel activo bajo. La tabla de función y el símbolo lógico de flip-flop se muestran en la figura 7-18.

entradas asincrónicas prestablecimiento borrado

Algunos flip-flops D tienen *entradas asincrónicas* que se pueden utilizar para llevar al flip-flop a un estado particular, independiente de las entradas CLK y D. Estas entradas, generalmente se identifican como PR (*present, prestablecimiento*) y CLR (*clear, borrado*), se comportan como las entradas de establecimiento y restablecimiento en un latch SR.

Figura 7-17 Comportamiento en el tiempo de un flip-flop D disparado por flanco positivo.



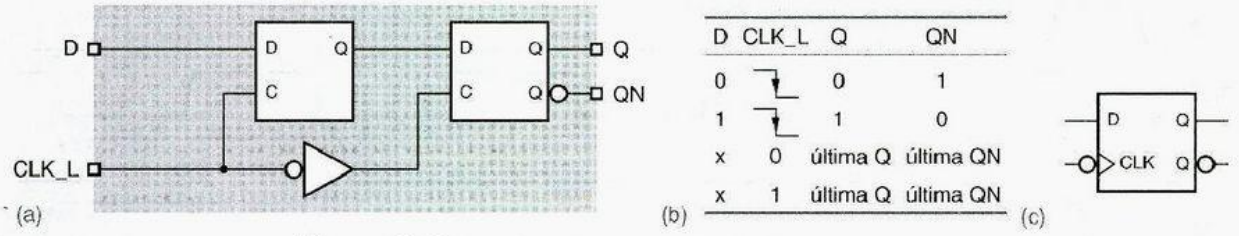


Figura 7-18 Flip-flop D disparado por flanco negativo: a) diseño de circuito empleando latchs D; b) tabla de función; c) símbolo lógico.

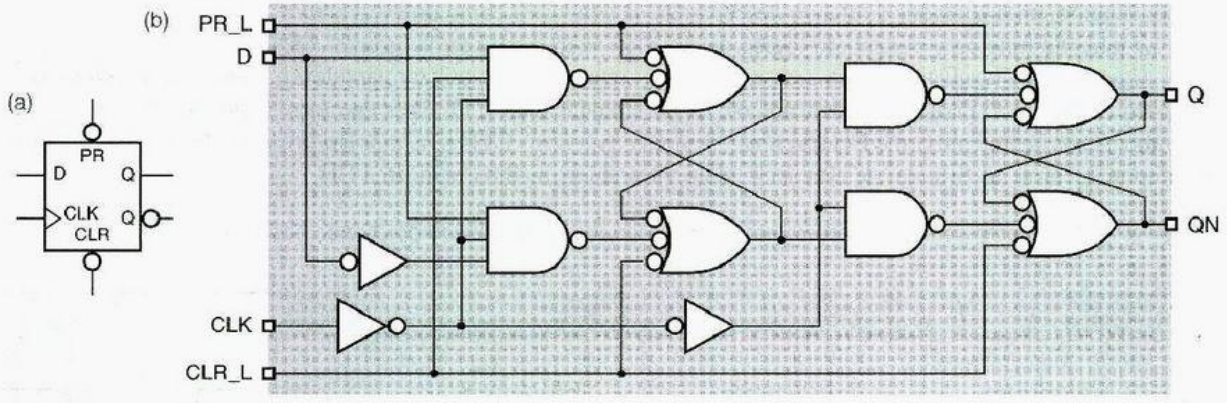
El símbolo lógico y el circuito NAND para un flip-flop D disparado pro flanco con estas entradas se ilustra en la figura 7-19. Aunque las entradas asincrónicas son utilizadas por algunos diseñadores lógicos para realizar funciones secuenciales difíciles, se reservan mejor para propósitos de inicialización y prueba, para forzar un circuito secuencial a un estado de inicio conocido; veremos más de esto cuando discutamos la metodología del diseño sincrónico en la sección 8.7.

7.2.6 Flip-flop D disparado por flanco con habilitación

entrada de habilitación
entrada de habilitación de reloj

Una función comúnmente deseada, en los flip-flops D es la habilidad para mantener el último valor almacenado, más que cargar un nuevo valor, durante el flanco de reloj. Esto se cumple al agregar una *entrada de habilitación*, llamada EN o CE (*clock enable, habilitación de reloj*). Mientras que el nombre de "habilitación de reloj" es descriptivo, la función de entrada extra no es obtenida en lo absoluto controlando el reloj de alguna manera. En vez de ello, como se

Figura 7-19 Flip-flop D disparado por flanco positivo con prestablecimiento y borrado; (a) símbolo lógico; (b) diseño del circuito que utiliza compuertas NAND.



TIEMPO PARA UN COMERCIAL

Los flip-flops D disparados por flanco positivo TTL comerciales no utilizan el diseño de latch maestro/esclavo de la figura 7-15 o la figura 7-19. En su lugar, los flip-flops como el 74LS74 emplean el diseño de seis compuertas de la figura 7-20, que es más pequeño y más rápido. Mostraremos cómo analizar formalmente el comportamiento de estado siguiente de ambos diseños en la sección 7.9.

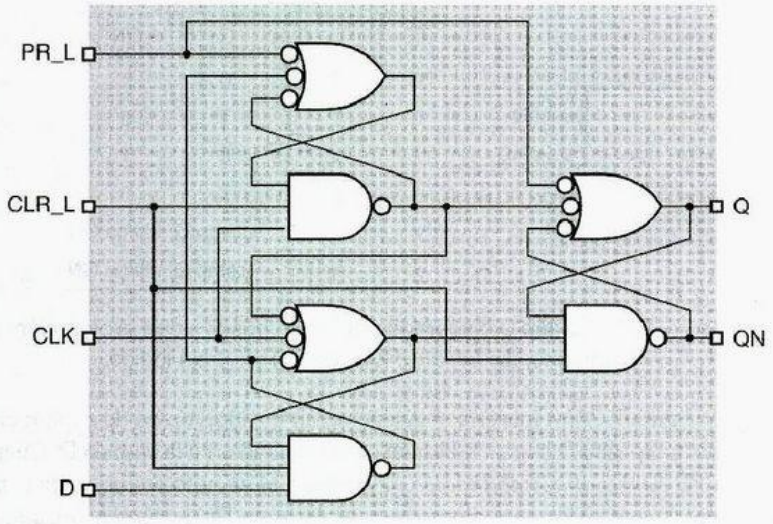


Figura 7-20
Circuito comercial para un flip-flop D disparado por flanco positivo como el 74LS74.

muestra en la figura 7-21(a), un multiplexor de 2 entradas controla el valor aplicado a la entrada D del flip-flop interno. Si EN es afirmada, la entrada D externa se selecciona; si EN es negada, se utiliza la salida actual del flip-flop. La tabla de función resultante se muestra en (b). El símbolo del flip-flop se ilustra en (c); en algunos flip-flops, la entrada de habilitación es de nivel activo bajo, denotada mediante una burbuja de inversión en esta entrada.

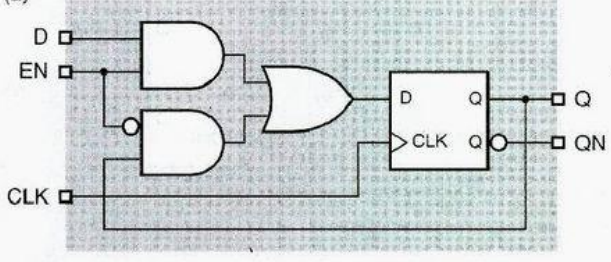
7.2.7 Flip-flop de exploración

capacidad de exploración

Una importante función del flip-flop para prueba ASIC es la conocida como *capacidad de exploración*. La idea es poder controlar la entrada D del flip-flop con una fuente alterna de datos durante la prueba del dispositivo. Cuando todos los flip-flops se ponen en modo de prueba, puede introducirse un patrón de prueba para el ASIC empleando las entradas de datos alternas de los flip-flops. Después de que se carga el patrón de prueba, los flip-flops se regresan al modo "normal" y todos los flip-flops se temporizan normalmente. Después de uno o más ciclos de reloj, los flip-flops se regresan al modo de prueba, y los resultados de la prueba son "explorados".

Figura 7-21 Flip-flop D disparado por flanco positivo con habilitación: (a) diseño del circuito; (b) tabla de función; (c) símbolo lógico.

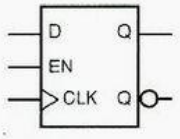
(a)

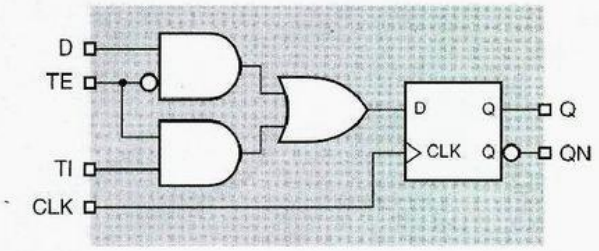


(b)

D	EN	CLK	Q	QN
0	1	↑	0	1
1	1	↑	1	0
x	0	↑	última Q	última QN
x	x	0	última Q	última QN
x	x	1	última Q	última QN

(c)





(a)

TE	TI	D	CLK	Q	QN
0	x	0	↑	0	1
0	x	1	↑	1	0
1	0	x	↑	0	1
1	1	x	↑	1	0
x	x	x	0	última Q	última QN
x	x	x	1	última Q	última QN

(b)

(c)

Figura 7-22 Flip-flop D disparado por flanco positivo con exploración: (a) diseño del circuito; (b) tabla de función; (c) símbolo lógico.

entrada de habilitación de prueba, TE

entrada de prueba, TI

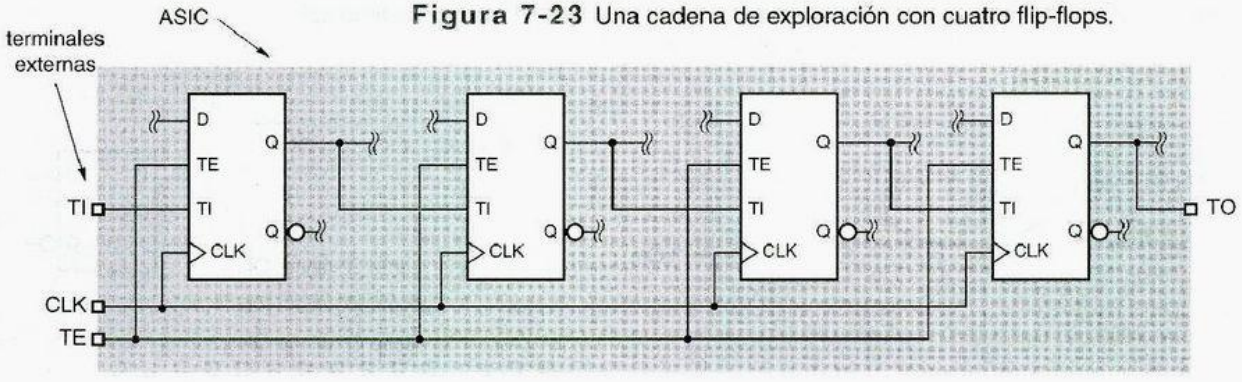
La figura 7-22(a) muestra el diseño de un flip-flop típico de exploración. No es otra cosa que un flip-flop D con un multiplexor de 2 entradas en la entrada D. Cuando se niega la entrada TE (*test enable, habilitación de prueba*), el circuito se comporta como un flip-flop ordinario. Cuando TE es asertiva, toma sus datos de TI (*test input, entrada de prueba*) en lugar de D. Este comportamiento funcional se muestra en (b) y se proporciona un símbolo para el dispositivo en (c).

cadena de exploración

Las entradas extra se utilizan para conectar la totalidad de los flip-flops de ASCI en una *cadena de exploración* para propósitos de prueba. La figura 7-23 es un ejemplo simple con cuatro flip-flops en la cadena de exploración. Las entradas TE de todos los flip-flops se encuentran conectadas a una entrada global TE, mientras que la salida Q de cada flip-flop está conectada a la entrada TI de otro de manera serial (en cadena). Las conexiones TI, TE y TO (test output, salida de prueba) son estrictamente para propósitos de prueba; la lógica adicional conectada a las entradas D y las salidas Q necesarias para hacer que el circuito haga algo útil no se ilustran.

Para probar el circuito, incluyendo la lógica adicional, la entrada TE global se mantiene asertiva mientras ocurren n ciclos de reloj y se aplican n bits de vector de prueba a la entrada TI global y por ello son barridos (desplazados) en los n flip-flops; n es igual a 4 en la figura 7-23. Luego TE es negado, y el circuito se deja correr por uno o más lazos de reloj adicionales. El nuevo estado del circuito, representado por los nuevos valores en los n flip-flops, pueden observarse (explorarse) en TO al afirmar TE mientras que ocurren n lazos de reloj más. Para hacer el proceso de prueba más eficiente, puede explorarse otro vector de prueba mientras que el resultado anterior se está explorando.

Figura 7-23 Una cadena de exploración con cuatro flip-flops.



Existen muchos tipos diferentes de flip-flops de exploración, correspondientes a diferentes tipos de funcionalidad de flip-flop básica. Por ejemplo, la capacidad de exploración podría agregarse al flip-flop D con habilitación de la figura 7-21 mediante el reemplazo de su multiplexor de 2 entradas interno con uno de 3 entradas. Para cada ciclo de reloj el flip-flop cargaría D, TI o su estado actual, dependiendo de los valores en EN y TE. La capacidad de exploración también puede agregarse a otros tipos de flip-flops, tales como los J-K y T que se presentan más adelante en esta sección.

*7.2.8 Flip-flop S-R maestro/esclavo

Indicamos con anterioridad que los latchs S-R son útiles en aplicaciones de "control", donde podemos tener condiciones independientes para establecer y restablecer un bit de control. Si se supone que el bit de control se cambia solamente en ciertos momentos con respecto a una señal de reloj, entonces necesitamos un flip-flop S-R que, como un flip-flop D, cambie sus salidas solamente en un cierto flanco de la señal de reloj. Esta subsección y las que le siguen describen los flip-flops que son útiles para tales aplicaciones.

flip-flop S–R maestro/esclavo

Si sustituimos los latchs S-R en lugar de los latchs D en el flip-flop D disparado por flanco negativo de la figura 7-18(a), obtenemos un *flip-flop S-R maestro/esclavo*, ilustrado en la figura 7-24. Como un flip-flop D, el flip-flop S-R cambia sus salidas solamente en el flanco descendente de una señal de control C. Sin embargo, el nuevo valor de salida depende de los valores de entrada no sólo en el flanco descendente, sino también durante el intervalo completo en el que C es 1 antes del flanco descendente. Como se muestra en la figura 7-25, un pulso breve sobre S en cualquier momento durante este intervalo puede establecer el latch maestro; igualmente, un pulso sobre R puede restablecerlo. El valor transferido a la salida del flip-flop en el flanco descendente de C depende de si el latch maestro fue finalmente establecido o borrado mientras que C era 1.

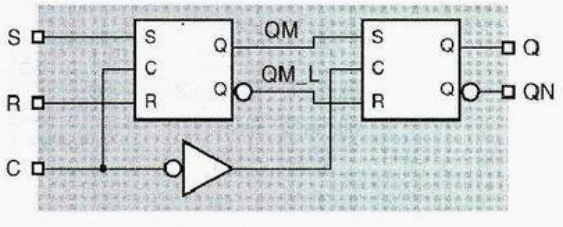
indicador de salida pospuesta

flip-flop disparado por pulso

Como se muestra en la figura 7-24(c), el símbolo lógico para el flip-flop S-R maestro/esclavo no utiliza un indicador de entrada dinámica, porque el flip-flop no es verdaderamente disparado por flanco. Es más como un latch que sigue su entrada durante todo el intervalo en que C es 1 pero cambia su salida para reflejar el valor final asegurado solamente cuando C tiende a 0. En el símbolo, un *indicador de salida pospuesta* indica que la señal de salida no cambia hasta que la entrada de habilitación de C sea negada. Los flip-flops con esta clase de comportamiento se conocen a menudo como *flip-flop disparados por pulso*.

Figura 7-24 Flip-flop S-R maestro/esclavo: a) circuito que utiliza latchs S-R; b) tabla de función; c) símbolo lógico.

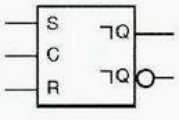
(a)



(b)

S	R	C	Q	QN
x	x	0	última Q	última QN
0	0	⎍	última Q	última QN
0	1	⎍	0	1
1	0	⎍	1	0
1	1	⎍	indef.	indef.

(c)



* En este libro, las secciones opcionales están marcadas con un asterisco.

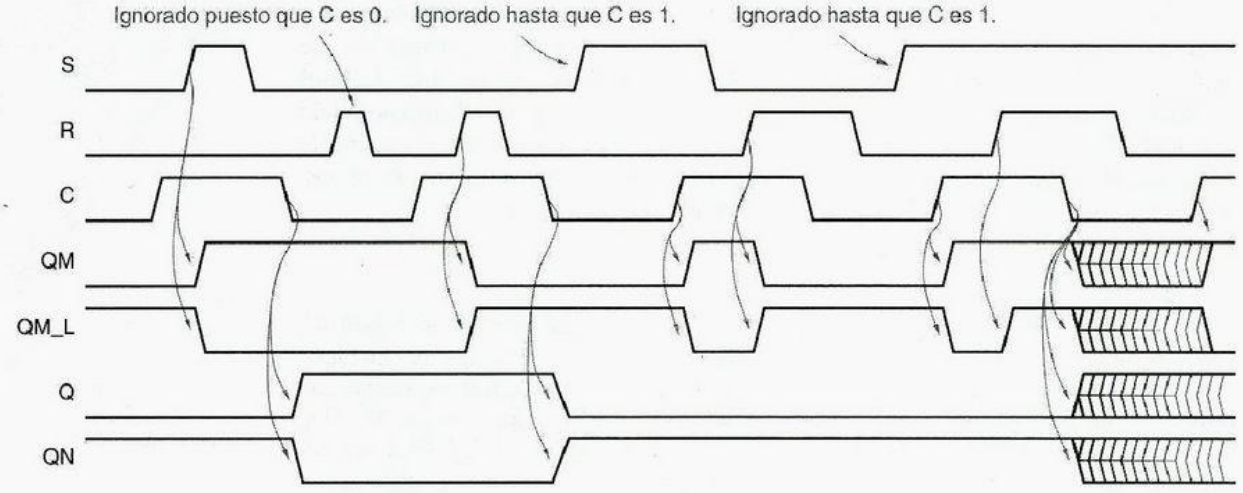


Figura 7-25 Comportamiento interno y funcional de un flip-flop S-R maestro/esclavo.

El funcionamiento del flip-flop S-R maestro/esclavo es imprevisible si tanto S como R son asertivas en el flanco descendente de C. En este caso, justo antes del flanco descendente, ambas salidas del latch maestro Q y QN son 1. Cuando C tiende a 0, las salidas del latch maestro cambian de manera impredecible e inclusive pueden llegar a ser metaestables. Al mismo tiempo, el latch maestro abre y propaga esta basura a la salida del flip-flop.

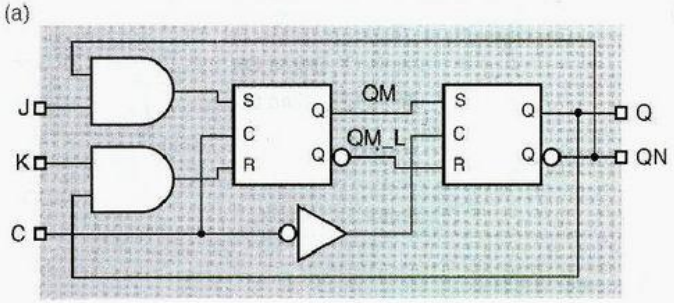
*7.2.9 Flip-flop J-K maestro/esclavo

flip-flop J-K maestro/esclavo

El problema de qué hacer cuando S y R se afirman de manera simultánea se resuelve en un *flip-flop J-K maestro/esclavo*. Las entradas J y K son análogas a S y R. Sin embargo, como se ilustra en la figura 7-26, afirmando J se afirma la entrada S del maestro sólo si la salida de QN del flip-flop es en ese momento 1 (es decir, Q es 0) y la afirmación de K afirma la entrada R del maestro solamente si Q es actualmente 1. De este modo, si J y K se afirman de modo simultáneo, el flip-flop se va al valor opuesto de su estado actual.

Figura 7-26 Flip-flop J-K maestro/esclavo: (a) diseño de circuito empleando latchs S-R; (b) tabla de función; (c) símbolo lógico.

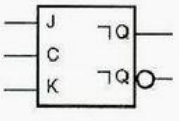
(a)



(b)

J	K	C	Q	QN
x	x	0	última Q	última QN
0	0	⎍	última Q	última QN
0	1	⎍	0	1
1	0	⎍	1	0
1	1	⎍	última QN	última Q

(c)



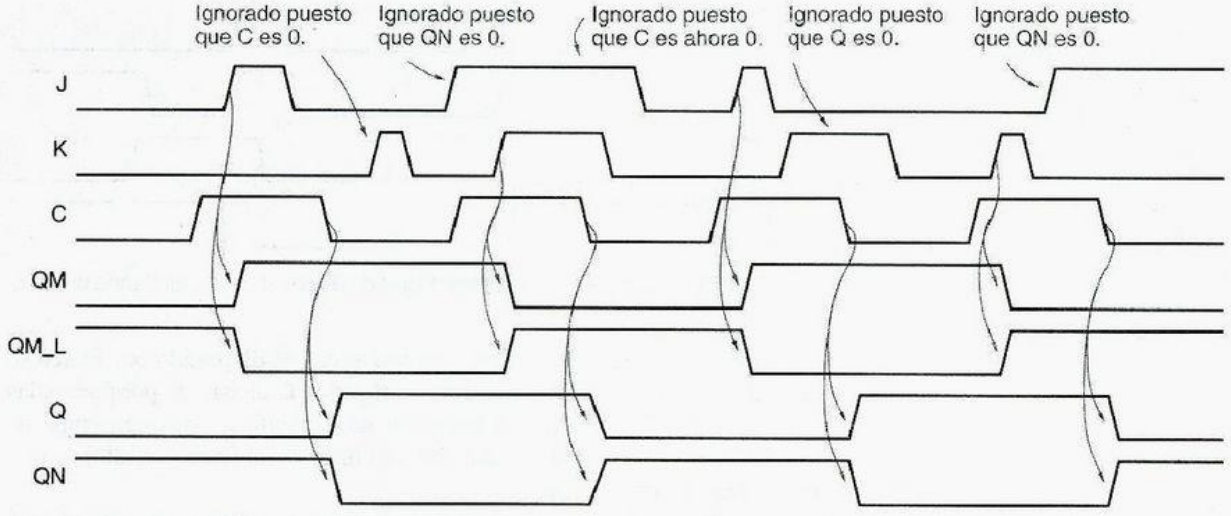


Figura 7-27 Comportamiento interno y funcional de un flip-flop J-K maestro/esclavo.

La figura 7-27 ilustra el comportamiento funcional de un flip-flop maestro/esclavo J-K para un conjunto típico de entradas. Note que las entradas J y K no necesitan ser afirmadas al final del pulso de disparo para que la salida del flip-flop cambie en ese momento. De hecho, debido a la conmutación en las entradas S y R del latch maestro, es posible para la salida del flip-flop cambiar a 1 aun cuando K y no J se afirme al final del pulso de disparo. Este comportamiento, conocido como *captura de unos*, se ilustra en el penúltimo pulso de disparo de la figura. Un comportamiento análogo conocido como captura de ceros se ilustra en el último pulso de disparo. Debido a este comportamiento, las entradas J y K de un flip-flop maestro/esclavo J-K debe mantenerse válido durante todo el intervalo en que C es 1.

captura de unos
captura de ceros

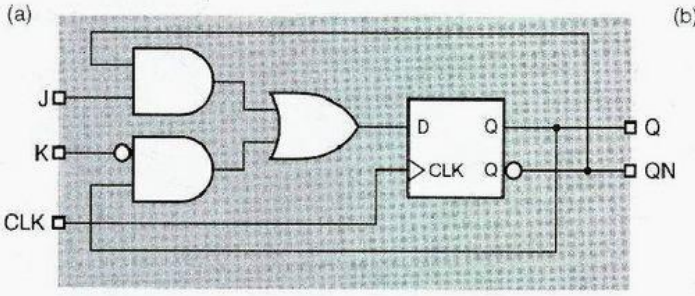
7.2.10 Flip-flop J-K disparado por flanco

El problema de la captura de unos y ceros se resuelve en un *flip-flop J-K disparado por flanco*, cuyo equivalente funcional se muestra en la figura 7-28. Al utilizar un flip-flop D disparado por flanco, internamente, el flip-flop J-K disparado por flanco muestrea sus entradas en el flanco ascendente del reloj y produce su siguiente salida de acuerdo con la "ecuación característica" $Q^* = J \cdot Q' + K' \cdot Q$ (véase la sección 7.3.3).

flip-flop J–K disparado por flanco

Figura 7-28 Flip-flop J-K disparado por flanco: (a) función equivalente que utiliza un flip-flop D disparado por flanco; (b) tabla de función; (c) símbolo lógico.

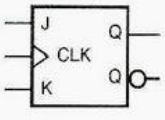
(a)



(b)

J	K	CLK	Q	QN
x	x	0	última Q	última QN
x	x	1	última Q	última QN
0	0	↓	última Q	última QN
0	1	↓	0	1
1	0	↓	1	0
1	1	↓	última QN	última Q

(c)



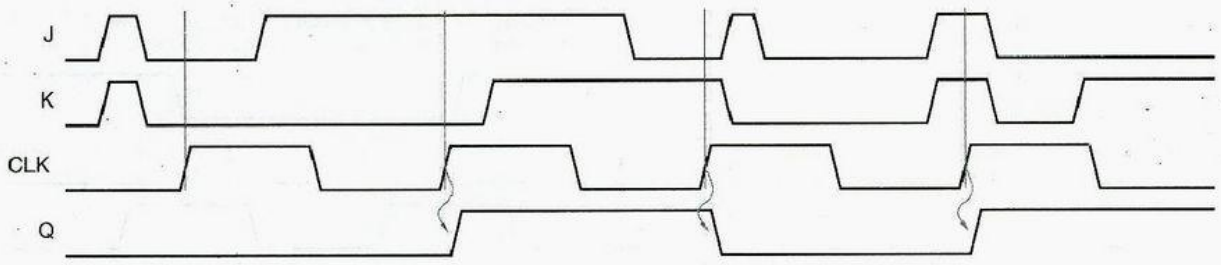


Figura 7-29 Comportamiento funcional de un flip-flop J-K disparado por flanco positivo.

El comportamiento funcional típico de un flip-flop J-K disparado por flanco se muestra en la figura 7-29. Como la entrada D de un flip-flop D disparado por flanco, las entradas J y K de un flip-flop J-K deben satisfacer las especificaciones de tiempo de establecimiento y tiempo de retención publicadas con respecto al flanco de reloj de disparo para un funcionamiento adecuado.

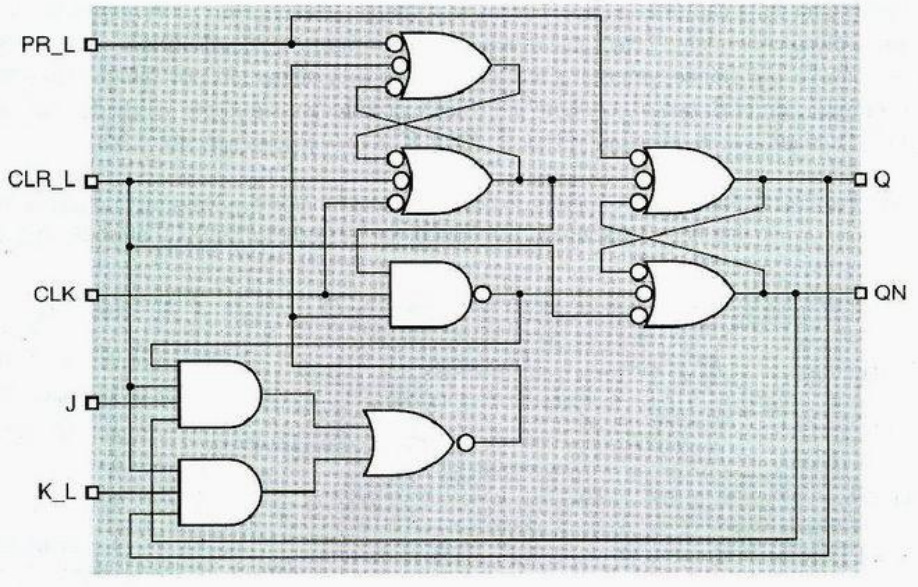
74x109

Debido a que eliminan los problemas de captura de unos y ceros y simultáneamente afirman ambas entradas de control, los flip-flops J-K disparados por flanco han hecho obsoletos en gran parte los antiguos tipos disparados por pulso. El *74x109* es un flip-flop TTL J-$\overline{K}$ disparado por flanco positivo con una entrada K de nivel activo bajo (llamada $\overline{K}$ o K_L).

OTRO COMERCIAL (FLIP-FLOP, ESO ES)

El diseño interno del 74LS109 es muy semejante al del 74LS74, el cual mostramos en la figura 7-20. Como se exhibe en la figura 7-30, el 109 simplemente reemplaza la compuerta inferior izquierda del 74, el cual realiza la ecuación característica $Q^* = D$, con una estructura AND-OR que realiza la ecuación característica J-$\overline{K}$, $Q^* = J \cdot Q' + K\_L \cdot Q$.

Figura 7-30
Diagrama lógico interno para el flip-flop J-K disparado por flanco positivo 74LS109.



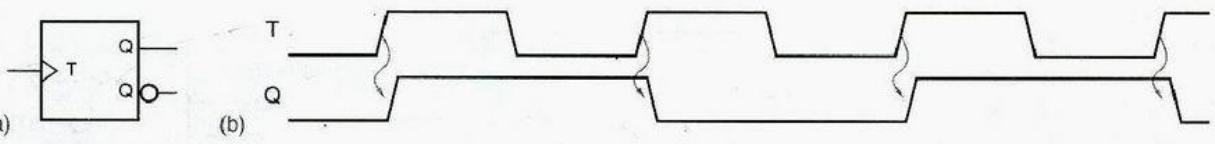


Figura 7-31 Flip-flop T disparado por flanco positivo: (a) símbolo lógico; (b) comportamiento funcional.

La aplicación más común de los flip-flops J-K se encuentra en las máquinas de estado sincrónicas temporizadas. Como explicamos en la sección 7.4.5, la lógica del estado siguiente para los flip-flops J-K es en ocasiones más sencilla que para los flip-flops D. Sin embargo, la mayoría de las máquinas de estado todavía están diseñadas empleando flip-flops D, porque la metodología del diseño es un poco más simple y porque la mayor parte de los dispositivos lógicos programables secuenciales contienen flip-flops D, no J-K. Por lo tanto, daremos mayor atención a los flip-flops D.

7.2.11 Flip-Flop T

flip-flop T

Un *flip-flop T (de "toggle", conmutación)* cambia de estado con cada pulso del reloj. La figura 7-31 muestra el símbolo e ilustra el comportamiento de un flip-flop T disparado por flanco positivo. Note que la señal en la salida Q del flip-flop tiene precisamente la mitad de la frecuencia de la entrada T. La figura 7-32 muestra cómo obtener un flip-flop T a partir de un flip-flop D o J-K. Los flip-flops T son utilizados con más frecuencia en contadores y divisores de frecuencia, como lo veremos en la sección 8.4.

flip-flop T con habilitación

En muchas aplicaciones de los flip-flops T, el flip-flop no necesita ser conmutado en cada pulso de reloj. Tales aplicaciones pueden utilizar un *flip-flop T con habilitación*. Como se ilustra en la figura 7-33, el flip-flop cambia su estado en el flanco de disparo del reloj solamente si la señal de habilitación EN está afirmada. Como las entradas D, J y K en otros flip-flops disparados por flanco, la entrada EN debe satisfacer los tiempos de establecimiento y retención especificados con respecto al flanco de reloj de disparo. Los circuitos de la figura 7-32 son fácilmente modificados para proporcionar una entrada EN, como se muestra en la figura 7-34.

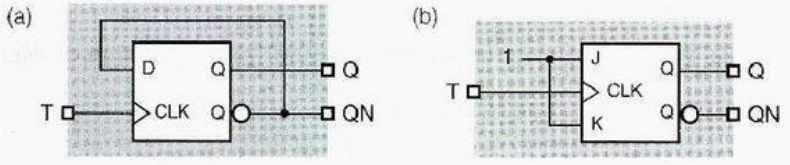
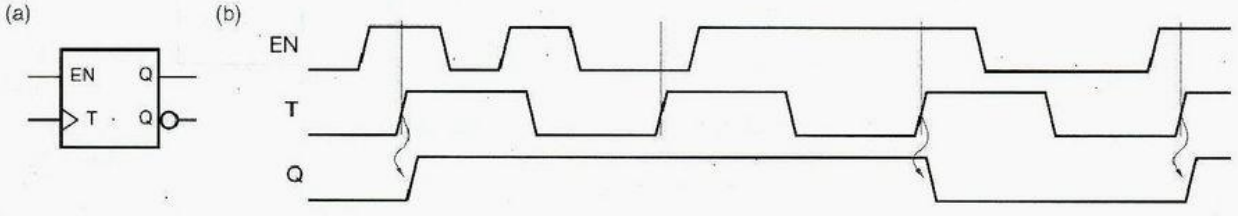


Figura 7-32
Posibles diseños de circuito para un flip-flop T: (a) que utiliza un flip-flop D; (b) que utiliza un flip-flop J-K.

Figura 7-33 Flip-flop T disparado por flanco positivo con habilitación: (a) símbolo lógico; (b) comportamiento funcional.



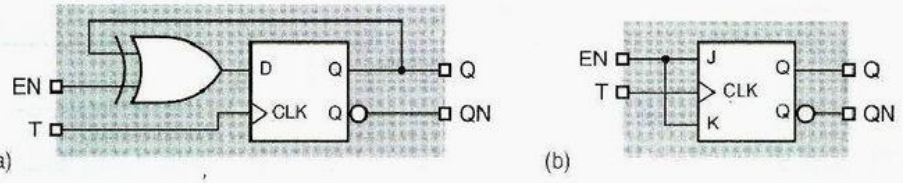


Figura 7-34
Posibles circuitos para un flip-flop T con habilitación: (a) que utiliza un flip-flop D; (b) empleando un flip-flop J-K.

7.3 Análisis de una máquina de estado sincrónica temporizada

máquina de estado sincrónica temporizada

Los latchs y flip-flops son los bloques de construcción básicos de los circuitos secuenciales, pero por sí solos son circuitos secuenciales de retroalimentación que se pueden analizar formalmente. Primero analizaremos el funcionamiento de las *máquinas de estado sincrónicas temporizadas*, puesto que son las más fáciles de comprender. "Máquina de estado" es un nombre genérico dado a estos circuitos secuenciales; "temporizada" hace referencia al hecho de que sus elementos de almacenamiento (flip-flops) emplean una entrada de reloj y "sincrónica" significa que todos los flip-flops utilizan la misma señal de reloj. Una máquina de estado de esta naturaleza cambia de estado solamente cuando se presenta un flanco de disparo o "pulso" en la señal de reloj.

7.3.1 Estructura de la máquina de estado

memoria de estado

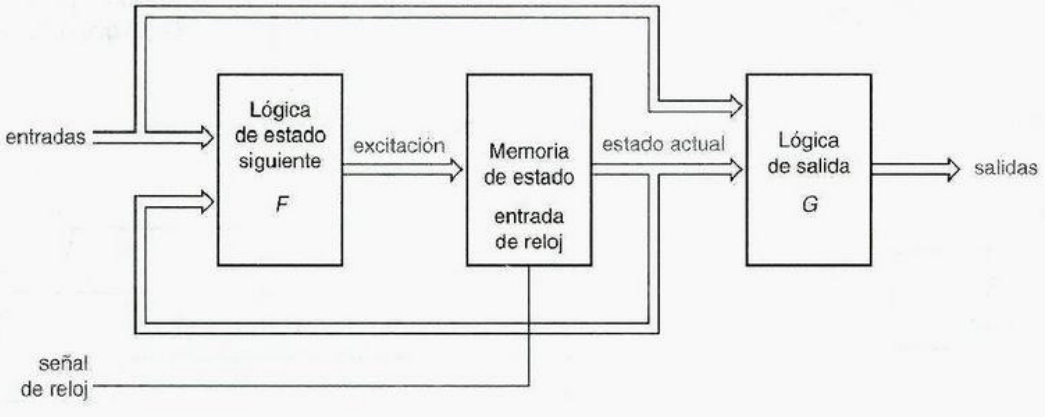
tic

La figura 7-35 ilustra la estructura general de una máquina de estado sincrónica temporizada. La *memoria de estado* es un conjunto de n flip-flops que almacenan el estado actual de la máquina, y tiene 2^n estados distintos. Los flip-flops se encuentran todos conectados a una señal de reloj común que ocasiona que cambien de estado en cada *pulso* o "*tic*" del reloj. Lo que constituya un "tic" depende del tipo de flip-flop (disparado por flanco, disparado por pulso, etc.). Para los flip-flops J-K y D disparados por flanco positivo que se consideran en esta sección, un "tic" es el flanco ascendente de la señal de reloj.

lógica de estado siguiente

El estado siguiente de la máquina de estado en la figura 7-35 está determinado por la *lógica de estado siguiente*, F como una función de la entrada y estado actuales.

Figura 7-35 Estructura de máquina de estado sincrónica temporizada (máquina de Mealy).



La *lógica de salida G* determina la salida como una función de la entrada y el estado actuales. Tanto *F* como *G* son estrictamente circuitos lógicos combinacionales. Podemos escribir

lógica de salida

$$\text{Estado siguiente} = F(\text{estado actual, entrada})$$
$$\text{Salida} = G(\text{estado actual, entrada})$$

Las máquinas de estado pueden emplear flip-flops D disparados por flanco positivo para su memoria de estado, en cuyo caso se presenta un "tic" en cada flanco ascendente de la señal de reloj. También es posible para la memoria de estado hacer uso de flip-flops D disparados por flanco negativo, latchs D o flip-flops J-K. Sin embargo, ya que la mayoría de las máquinas de estado se diseñan en la actualidad con la ayuda de dispositivos de lógica programables con flip-flops D disparados por flanco positivo, en ello nos concentraremos.

7.3.2 Lógica de salida

Un circuito secuencial cuya salida depende tanto del estado como de la entrada como el que se muestra en la figura 7-35 se conoce como una *máquina de Mealy*. En algunos circuitos secuenciales la salida depende solamente del estado:

$$\text{Salida} = G(\text{estado actual})$$

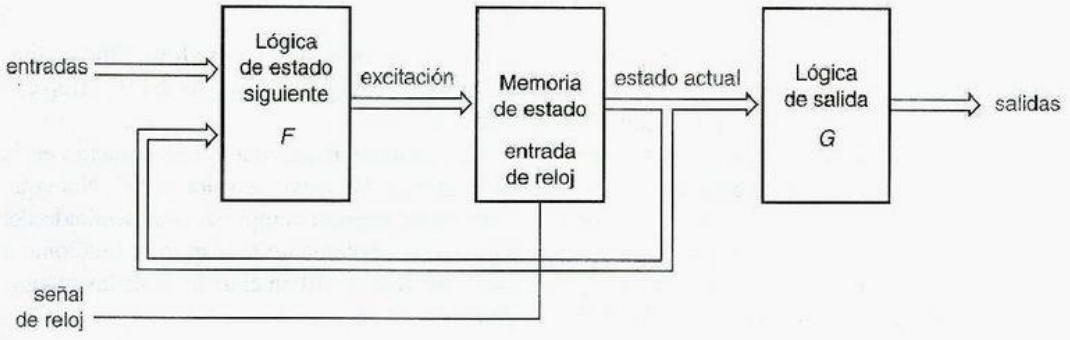
Un circuito así se conoce como una *máquina de Moore*, y su estructura general se muestra en la figura 7-36.

Obviamente, la única diferencia entre los dos modelos de máquina de estado se encuentra en cómo son generadas las salidas. En la práctica, muchas máquinas de estado deben ser categorizadas como máquinas de Mealy, porque tienen una o más *salidas del tipo Mealy* que dependen de la entrada así como también del estado. Sin embargo, muchas de estas máquinas también tienen una o más *salidas de tipo Moore* que dependen solamente del estado.

En el diseño de circuitos de alta velocidad, con frecuencia, es necesario asegurar que las salidas de la máquina de estado se encuentren disponibles tan pronto como sea posible y que no cambien durante cada periodo de reloj. Una forma de conseguir este comportamiento es codificar el estado de modo que las mismas variables de estado sirvan como salidas. Llamamos a esto una *asignación de estado codificada por salida*; produce una máquina de Moore en la cual la lógica de salida de la figura 7-36 es nula, compuesta solamente de alambres.

asignación de estado codificada por salida

Figura 7-36 Estructura de máquina de estado sincrónica temporizada (máquina de Moore).



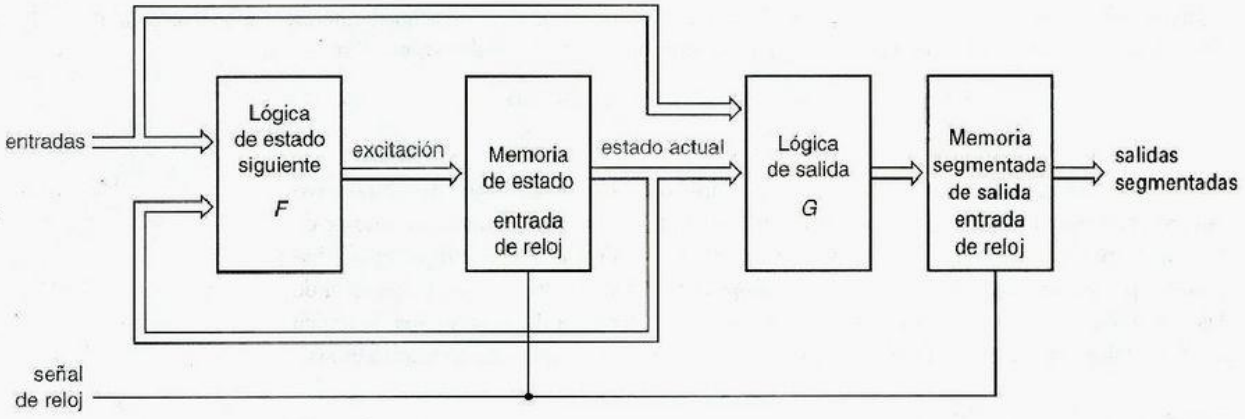


Figura 7-37 Máquina de Mealy con salidas segmentadas.

salidas segmentadas

Otro enfoque es diseñar la máquina de estado de manera que las salidas durante un periodo de reloj dependan del estado y las entradas durante el *anterior* periodo de reloj. Llamamos a estas *salidas segmentadas (pipelined outputs)*, y se obtienen al unir otra etapa de memoria (flip-flops) a las salidas de la máquina de Mealy como en la figura 7-37.

Con manipulaciones de circuito apropiado, se puede mapear un modelo de máquina de estado en otro. Por ejemplo, se podría declarar los flip-flops que producen salidas segmentadas de una máquina de Mealy como siendo parte de su memoria de estado y por ello obtener una máquina de Moore con una asignación de estado codificada por salida.

La clasificación exacta de una máquina de estado en un estilo u otro no es tan importante. Lo que es importante es cómo se piense acerca de la estructura de salida y cómo se satisfagan sus objetivos globales de diseño, incluyendo temporización y flexibilidad. Por ejemplo, las salidas segmentadas son excelentes para temporización, pero usted puede utilizarlas sólo en situaciones donde se puede averiguar el siguiente valor de salida deseado con un adelanto de un periodo de reloj. En cualquier aplicación dada usted puede emplear diferentes estilos para diferentes señales de salida. Por ejemplo, veremos en la sección 7.11.5 que pueden utilizarse diferentes instrucciones para especificar diferentes estilos de salida en ABEL.

7.3.3 Ecuaciones características

ecuación característica

El comportamiento funcional de un latch o flip-flop puede describirse formalmente mediante una *ecuación característica* que especifique el siguiente estado del flip-flop como una función de su estado y entradas actuales.

*sufijo **

Las ecuaciones características de los flip-flops de la sección 7.2 se numeran en la tabla 7-1. Por convención, el sufijo * en Q* significa "el siguiente valor de Q". Note que la ecuación característica no describe el comportamiento de temporización detallado del dispositivo (de latch vs. disparado por flanco, etc.), solamente la respuesta funcional a las entradas de control. Esta descripción simplificada es útil en el análisis de las máquinas de estado, como lo demostraremos dentro de poco.

Tipo de dispositivo	*Ecuación característica*
Latch S-R	$Q^* = S + R' \cdot Q$
Latch	$Q^* = D$
Flip-flop D disparado por flanco	$Q^* = D$
Flip-flop D con habilitación	$Q^* = EN \cdot D + EN' \cdot Q$
Flip-flop S-R maestro/esclavo	$Q^* = S + R' \cdot Q$
Flip-flop J-K maestro/esclavo	$Q^* = J \cdot Q' + K' \cdot Q$
Flip-flop J-K disparado por flanco	$Q^* = J \cdot Q' + K' \cdot Q$
Flip-flop T	$Q^* = Q'$
Flip-flop T con habilitación	$Q^* = EN \cdot Q' + EN' \cdot Q$

Tabla 7-1
Ecuaciones características de latch y flip-flop.

7.3.4 Análisis de máquinas de estado con flip-flops D

Considere la definición formal de una máquina de estado que dimos con anterioridad:

$$\text{Estado siguiente} = F(\text{estado actual, entrada})$$
$$\text{Salida} = G(\text{estado actual, entrada})$$

Recordando nuestra noción de que "estado" incorpora todo lo que necesitamos saber acerca de la historia pasada del circuito, la primera ecuación nos dice que lo que necesitamos saber a continuación puede ser determinado a partir de lo que actualmente sabemos y de la entrada actual. La segunda ecuación nos dice que la salida actual puede ser determinada a partir de la misma información. El objetivo del análisis de circuito secuencial es determinar las funciones de salida y estado siguiente de modo que pueda predecirse el comportamiento de un circuito.

El análisis de una máquina de estado sincrónica temporizada tiene tres pasos básicos:

1. Determinar las funciones de estado siguiente y de salida F y G.
2. Utilizar F y G para construir una *tabla de estado/salida* que especifique completamente el estado siguiente y la salida del circuito para toda posible combinación de estado actual y salida. *tabla de estado/salida*
3. (Opcional) Dibujar un *diagrama de estado* que presente la información de los pasos anteriores en forma gráfica. *diagrama de estado*

La figura 7-38 muestra una máquina de estado simple con dos flip-flops D disparados por flanco positivo. Para determinar la función de estado siguiente F, primero debemos considerar el comportamiento de la memoria de estado. En el flanco ascendente de la señal de reloj, cada flip-flop D muestrea su entrada D y transfiere este valor a su salida Q; la ecuación característica de un flip-flop D es $Q^* = D$. Por lo tanto, para determinar el siguiente valor de Q (es decir, Q*), primero debemos determinar el valor actual de D.

En la figura 7-38 tenemos dos flip-flops D, y hemos nombrado las señales en sus salidas como Q0 y Q1. Estas dos salidas son las variables de estado; su valor es el estado

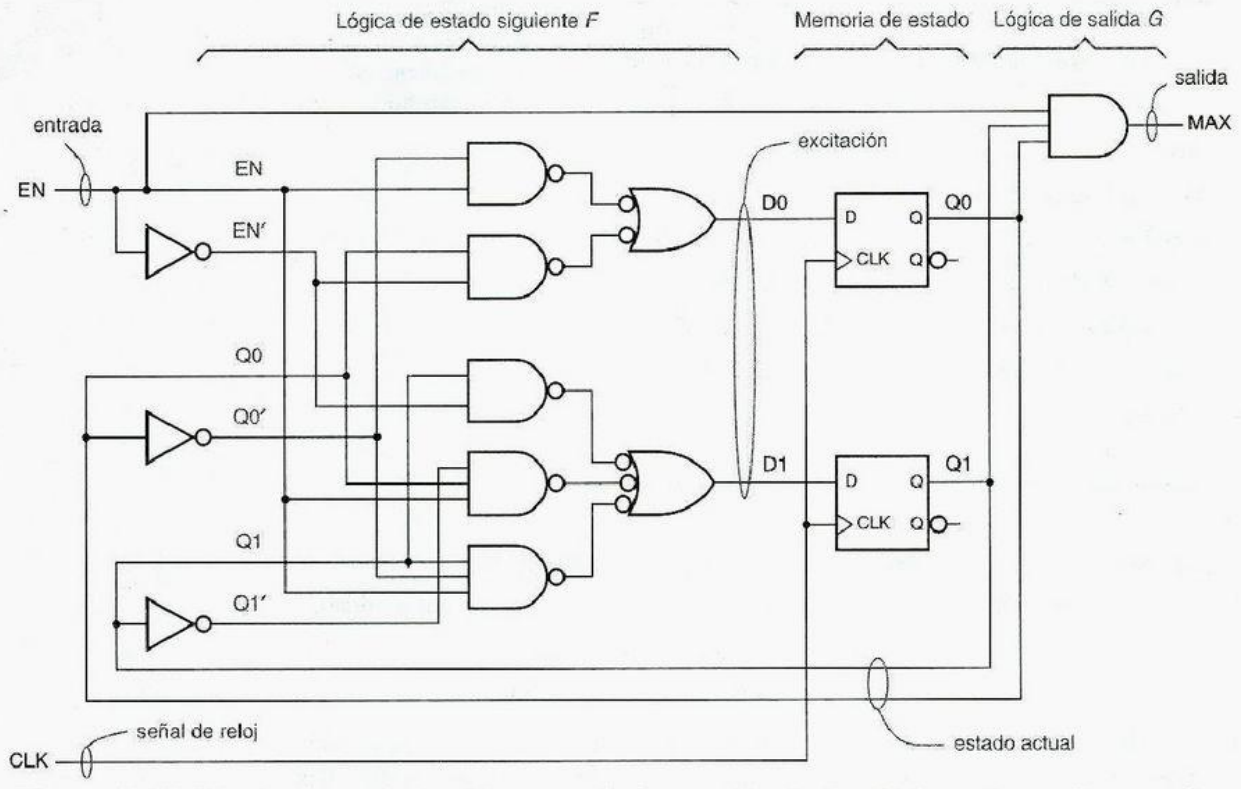


Figura 7-38 Máquina de estado sincrónica temporizada que utiliza flip-flops D disparados por flanco positivo.

excitación

ecuación de excitación

actual de la máquina. Hemos nombrado las señales en las entradas D correspondientes como D0 y D1. Estas señales proporcionan la *excitación* para los flip-flops D en cada tic de reloj. Las ecuaciones lógicas que expresan las señales de excitación como funciones del estado y entrada actuales se denominan *ecuaciones de excitación* y pueden obtenerse del diagrama del circuito:

$$D0 = Q0 \cdot EN' + Q0' \cdot EN$$

$$D1 = Q1 \cdot EN' + Q1' \cdot Q0 \cdot EN + Q1 \cdot Q0' \cdot EN$$

*sufijo **

Por convención, el valor siguiente de una variable de estado después de un tic de reloj se denota agregando un asterisco al nombre de la variable de estado, por ejemplo, Q0* o Q1*. Al aplicar la ecuación característica de los flip-flops D, Q* = D, podemos describir la función del estado siguiente de la máquina de ejemplo con ecuaciones para el valor siguiente de las variables de estado:

$$Q0* = D0$$

$$Q1* = D1$$

Sustituyendo las ecuaciones de excitación para D0 y D1, podemos escribir

$$Q0* = Q0 \cdot EN' + Q0' \cdot EN$$

$$Q1* = Q1 \cdot EN' + Q1' \cdot Q0 \cdot EN + Q1 \cdot Q0' \cdot EN$$

(a)

	EN	
Q1 Q0	0	1
00	00	01
01	01	10
10	10	11
11	11	00
	Q1* Q0*	

(b)

	EN	
S	0	1
A	A	B
B	B	C
C	C	D
D	D	A
	S*	

(c)

	EN	
S	0	1
A	A, 0	B, 0
B	B, 0	C, 0
C	C, 0	D, 0
D	D, 0	A, 1
	S*, MAX	

Tabla 7-2
Tablas de transición, estado y estado/salida para la máquina de estado en la figura 7-38.

Estas ecuaciones, que expresan el valor siguiente de las variables de estado como una función del estado y entrada actuales, se denominan *ecuaciones de transición.* *ecuación de transición*

Para cada combinación de valor de entrada y estado actual, las ecuaciones de transición predicen el estado siguiente. Cada estado se encuentra descrito por dos bits, los valores actuales de Q0 y Q1: (Q1 Q0) = 00, 01, 10 o 11. [La razón para elegir "arbitrariamente" el orden (Q1 Q0) en lugar de (Q0 Q1) se hará evidente en breve.] Para cada estado, nuestra máquina de ejemplo solamente tiene dos valores posibles de entrada, EN = 0 o EN = 1, de manera que hay un total de 8 combinaciones de estado/entrada (En general, una máquina con s bits de estado e i entradas tiene 2^{s+i} combinaciones de estado/entrada.)

La tabla 7-2(a) muestra una *tabla de transición* que es creada al evaluar las ecuaciones de transición para cada posible combinación de estado/entrada. Tradicionalmente, una tabla de transición numera los estados en la izquierda y las combinaciones de entrada en la parte superior de la tabla, como se muestra en el ejemplo. *tabla de transición*

La función de nuestra máquina de ejemplo es evidente de su tabla de transición: es un contador binario de 2 bits con una entrada de habilitación EN. Cuando EN = 0, la máquina mantiene su conteo actual, pero cuando EN = 1, el conteo se adelanta en 1 en cada tic de reloj, regresando a 00 cuando alcanza un valor máximo de 11.

Si deseamos, podemos asignar *nombres de estado* alfanuméricos a cada estado. La asignación más simple es 00 = A, 01 = B, 10 = C y 11 = D. Sustituyendo los nombres de estado para las combinaciones de Q1 y Q0 (y Q1* y Q0*) en la tabla 7-2(a) se produce la *tabla de estado* en (b). Aquí, "S" denota el estado actual y "S*" denota el estado siguiente de la máquina. Una tabla de estado es por lo regular más fácil de comprender que una tabla de transición, porque en máquinas complejas podemos utilizar nombres de estado que tienen significado. Sin embargo, una tabla de estado contiene menos información que una tabla de transición porque no indica los valores binarios asumidos por las variables de estado en cada estado nombrado. *nombres de estado* *tabla de estado*

Una vez que se produce una tabla de estado, solamente queda la lógica de salida de la máquina para analizar. En la máquina de ejemplo sólo hay una sola señal de salida, y es una función tanto del estado como de la entrada actuales (ésta es una máquina de Mealy). De modo que podemos escribir una *ecuación de salida* simple: *ecuación de salida*

$$MAX = Q1 \cdot Q0 \cdot EN$$

El comportamiento de la salida que se predice con esta ecuación puede combinarse con la información del estado siguiente para producir una *tabla de estado/salida* como se muestra en la tabla 7-2(c). *tabla de estado/salida*

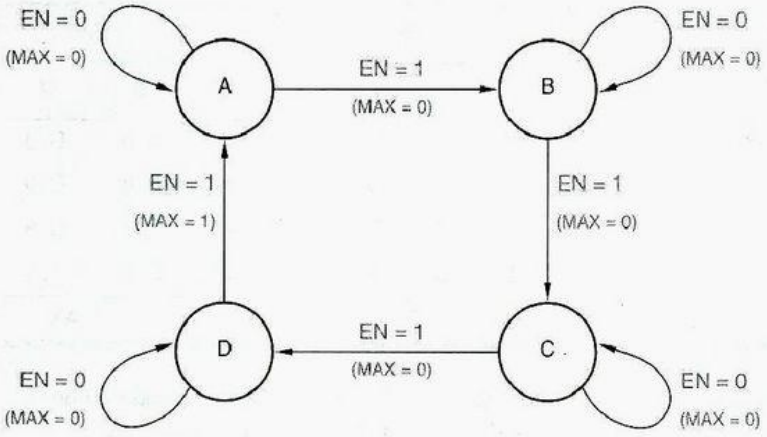


Figura 7-39
Diagrama de estado correspondiente a la máquina de estado de la tabla 7-2.

Las tablas de estado/salida para máquinas de Moore son ligeramente más simples. Por ejemplo, en el circuito de la figura 7-38 supongamos que retiramos la señal EN de la compuerta AND que produce la salida MAX, produciendo una salida tipo Moore MAXS. Entonces MAXS es una función solamente del estado, y la tabla estado/salida puede enumerar MAXS en una sola columna, independiente de los valores de entrada. Esto se muestra en la tabla 7-3.

diagrama de estado
nodo
arco dirigido

Un *diagrama de estado* presenta la información de la tabla de estado/salida en un formato gráfico. Tiene un círculo (o *nodo*) para cada estado y una flecha (o *arco dirigido*) para cada transición. La figura 7-39 muestra el diagrama de estado para nuestra máquina de estado de ejemplo. La letra dentro de cada círculo es un nombre de estado. Cada flecha que abandona un estado dado apunta al siguiente estado para una combinación de entrada dada; también muestra el valor de salida producido en el estado dado para esa combinación de entrada.

El diagrama de estado para una máquina de Moore puede ser algo más simple. En este caso, los valores de salida pueden mostrarse dentro de cada círculo de estado, puesto

Tabla 7-3
Tabla de estado/salida para una máquina de Moore.

	EN		
S	***0***	***1***	***MAXS***
A	A	B	0
B	B	C	0
C	C	D	0
D	D	A	1
	S*		

UNA ACLARACIÓN La notación del diagrama de estado para valores de salida en máquinas de Mealy es un poco engañosa. Usted debería recordar que el valor de salida enunciado se produce continuamente cuando la máquina se encuentra en el estado indicado y tiene la entrada indicada, no sólo durante la transición al estado siguiente.

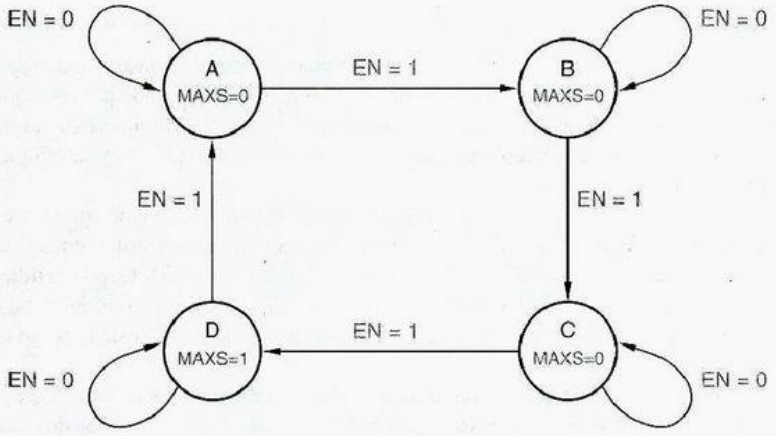
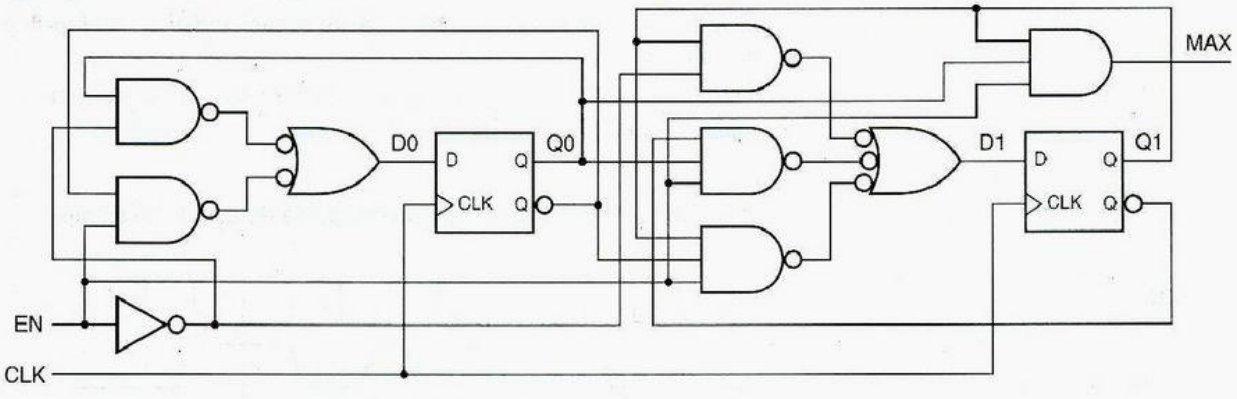


Figura 7-40
Diagrama de estado correspondiente a la máquina de estado de la tabla 7-3.

que son funciones de estado solamente. El diagrama de estado para una máquina de Moore que emplea esta convención se muestra en la figura 7-40.

El diagrama lógico original de nuestra máquina de estado de ejemplo, figura 7-38, fue diseñado para satisfacer nuestro modelo conceptual de una máquina de Mealy. Sin embargo, nada nos exige agrupar la lógica del estado siguiente, la memoria de estado y la lógica de salida de esta manera. La figura 7-41 muestra otro diagrama lógico para la misma máquina de estado. Para analizar este circuito, el diseñador (o analizador, en este caso) todavía puede extraer la información requerida del diagrama como está dibujado.

Figura 7-41 Trazando de nuevo el diagrama lógico para una máquina de estado sincrónica temporizada.



FLECHAS PEQUEÑAS, FLECHAS PEQUEÑAS POR TODAS PARTES

Puesto que hay solamente una entrada en nuestra máquina de ejemplo, sólo existen dos posibles combinaciones de entrada, y dos flechas dejando cada estado. En una máquina con n entradas, tendríamos 2^n flechas dejando cada estado. Esto es complicado si n es grande. Posteriormente, en la figura 7-44, describiremos una convención por la cual un estado no necesita tener una flecha saliendo de él para cada combinación de entrada, solamente una flecha para cada distinto estado siguiente.

DIBUJOS SUGERENTES

Si utilizamos las tablas de transición, estado y salida, podemos construir un diagrama de temporización que muestre el comportamiento de una máquina de estado para cualquier estado de inicio deseado y secuencia de entrada. Por ejemplo, la figura 7-42 muestra el comportamiento de nuestra máquina de ejemplo con un estado de inicio de 00 (A) y un patrón particular en la entrada EN.

Nótese que el valor de la entrada EN afecta al estado siguiente solamente en el flanco ascendente de la entrada CLOCK; es decir, el contador cuenta solamente si EN = 1 en el flanco ascendente de CLOCK. Por otra parte, puesto que MAX es una salida tipo Mealy, su valor es afectado por EN todo el tiempo. Si también proporcionamos una salida MAXS tipo Moore como se sugiere en el texto, su valor depende solamente del estado como se muestra en la figura.

El diagrama de temporización se dibuja de una manera que muestra los cambios en las salidas MAX y MAXS que ocurren ligeramente después que los cambios de estado y entrada que los causan, reflejando el retardo de lógica combinacional de los circuitos de salida. Naturalmente, los dibujos son meramente sugerentes; la temporización precisa se indica normalmente mediante una tabla de tiempos del tipo sugerido en la sección 5.2.1.

La única diferencia del circuito en el nuevo diagrama es que hemos empleado las salidas QN de los flip-flops (las cuales son normalmente el complemento de Q) para ahorrar un par de inversores.

En resumen, los pasos detallados para analizar una máquina de estado sincrónica temporizada se numeran a continuación:

ecuaciones de excitación

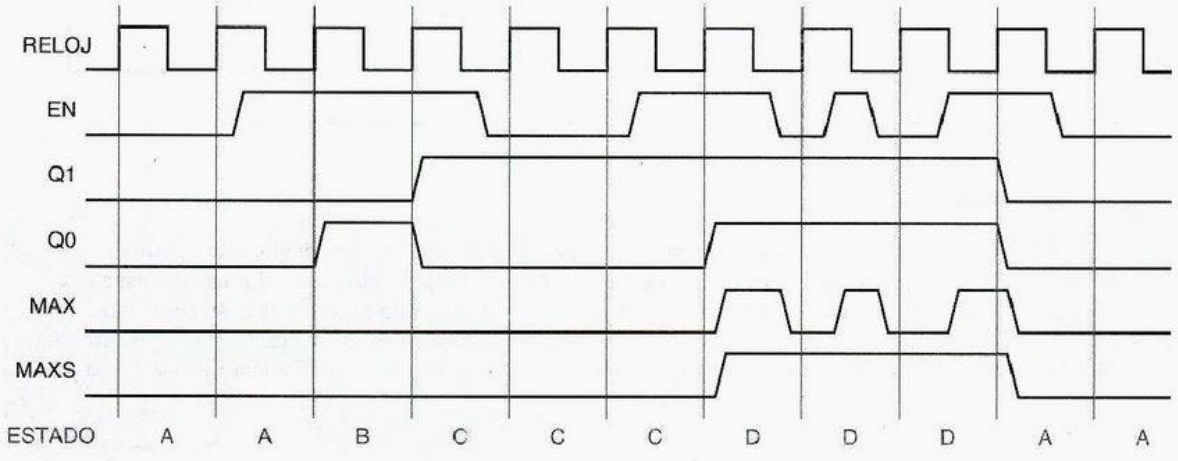
ecuaciones de transición

tabla de transición

ecuaciones de salida

1. Determinar las ecuaciones de excitación para las entradas de control del flip-flop.
2. Sustituir las ecuaciones de excitación en las ecuaciones características del flip-flop para obtener ecuaciones de transición.
3. Hacer uso de las ecuaciones de transición para construir una tabla de transición.
4. Determinar las ecuaciones de salida.

Figura 7-42 Diagrama de temporización para la máquina de estado de ejemplo.



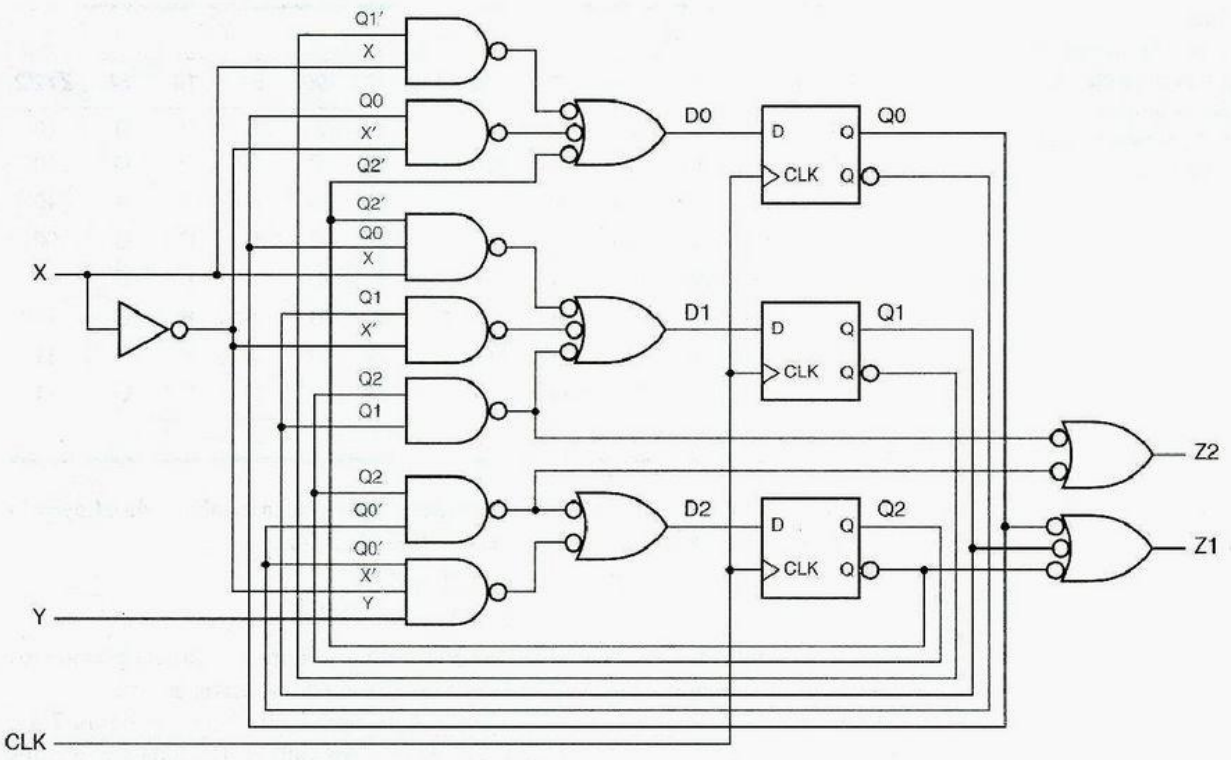


Figura 7-43 Una máquina de estado sincrónica temporizada con tres flip-flops y ocho estados.

5. Agregar los valores de salida a la tabla de transición para cada combinación de estado (Moore) o de estado/entrada (Mealy) para crear una tabla de transición/salida. *tabla de transición/salida*
6. Nombrar los estados y sustituir nombres de estado por combinaciones de variables de estado en la tabla de transición/salida para obtener una tabla de estado/salida. *nombres de estado* *tabla de estado/salida*
7. (Opcional) Dibujar un diagrama de estado correspondiente a la tabla de estado/salida. *diagrama de estado*

Pasaremos a través de toda esta secuencia de pasos para analizar otra máquina de estado sincrónica temporizada, ilustrada en la figura 7-43. Leyendo el diagrama lógico, encontramos que las ecuaciones de excitación son las siguientes:

$$D0 = Q1' \cdot X + Q0 \cdot X' + Q2$$
$$D1 = Q2' \cdot Q0 \cdot X + Q1 \cdot X' + Q2 \cdot Q1$$
$$D2 = Q2 \cdot Q0' + Q0' \cdot X' \cdot Y$$

Sustituyendo en la ecuación característica para flip-flops D, obtenemos las ecuaciones de transición:

$$Q0* = Q1' \cdot X + Q0 \cdot X' + Q2$$
$$Q1* = Q2' \cdot Q0 \cdot X + Q1 \cdot X' + Q2 \cdot Q1$$
$$Q2* = Q2 \cdot Q0' + Q0' \cdot X' \cdot Y$$

Tabla 7-4
Tablas de transición/salida y de estado/salida para la máquina de estado de la figura 7-43.

(a)

	XY				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	Q2* Q1* Q0*				

(b)

	XY				
S	00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S*				

Una tabla de transición basada en estas ecuaciones se muestra en la tabla 7-4a). Leyendo el diagrama lógico, podemos escribir dos ecuaciones de salida:

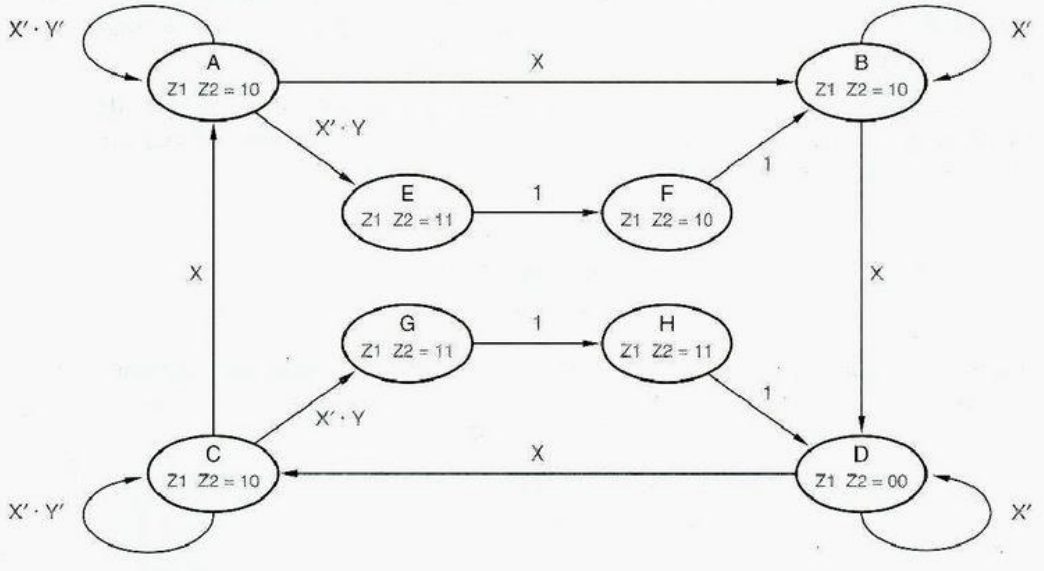
$$Z1 = Q2 + Q1' + Q0'$$
$$Z2 = Q2 \cdot Q1 + Q2 \cdot Q0'$$

Los valores de salida resultantes se muestran en la última columna de (a). Asignando los nombres de estado A-H, obtenemos la tabla de estado/salida mostrada en (b).

Un diagrama de estado para la máquina de ejemplo se ilustra en la figura 7-44. Puesto que nuestro ejemplo es una máquina de Moore, los valores de salida son escritos con cada estado. Cada arco se etiqueta con una *expresión de transición*; se toma una transición para combinaciones de entrada para las cuales la expresión de transición es 1. Las transiciones etiquetadas "1" siempre son tomadas.

expresión de transición

Figura 7-44 Diagrama de estado correspondiente a la tabla 7-4.



Las expresiones de transición en los arcos que abandonan un estado particular deben ser mutuamente exclusivas y todas inclusivas, como se explica a continuación:

- Ningún par de expresiones de transición puede ser igual a 1 para la misma combinación de entrada, puesto que una máquina no puede tener dos estados siguientes para una combinación de entrada. *exclusión mutua*
- Para toda posible combinación de entrada, alguna expresión de transición debe ser igual a 1, de modo que todos los estados siguientes están definidos. *inclusión total*

Comenzando con la tabla de estado, una expresión de transición para un estado actual y estado siguiente en particular puede escribirse como una suma de minitérminos para las combinaciones de entrada que causan esa transición. Si se desea, la expresión puede ser entonces minimizada para proporcionar la información en una forma más compacta. Las expresiones de transición son más útiles en el *diseño* de máquinas de estado, donde las expresiones se pueden desarrollar a partir de la descripción en palabras del problema, como se mostrará en la sección 7.5.

*7.3.5 Análisis de máquinas de estado con flip-flops J-K

Las máquinas de estado sincrónicas temporizadas que se construyen a partir de flip-flops J-K también se pueden analizar aplicando el procedimiento básico que se indicó en la subsección anterior. La única diferencia es que existen dos ecuaciones de excitación para cada flip-flop: una para J y la otra para K. Para obtener las ecuaciones de transición, ambas deben sustituirse en la ecuación característica de J-K, $Q^* = J \cdot Q' + K' \cdot Q$.

La figura 7-45 es una máquina de estado que utiliza flip-flops J-K. Leyendo el diagrama lógico, podemos obtener las siguientes ecuaciones de excitación:

$$J0 = X \cdot Y'$$
$$K0 = X \cdot Y' + Y \cdot Q1$$
$$J1 = X \cdot Q0 + Y$$
$$K1 = Y \cdot Q0' + X \cdot Y' \cdot Q0$$

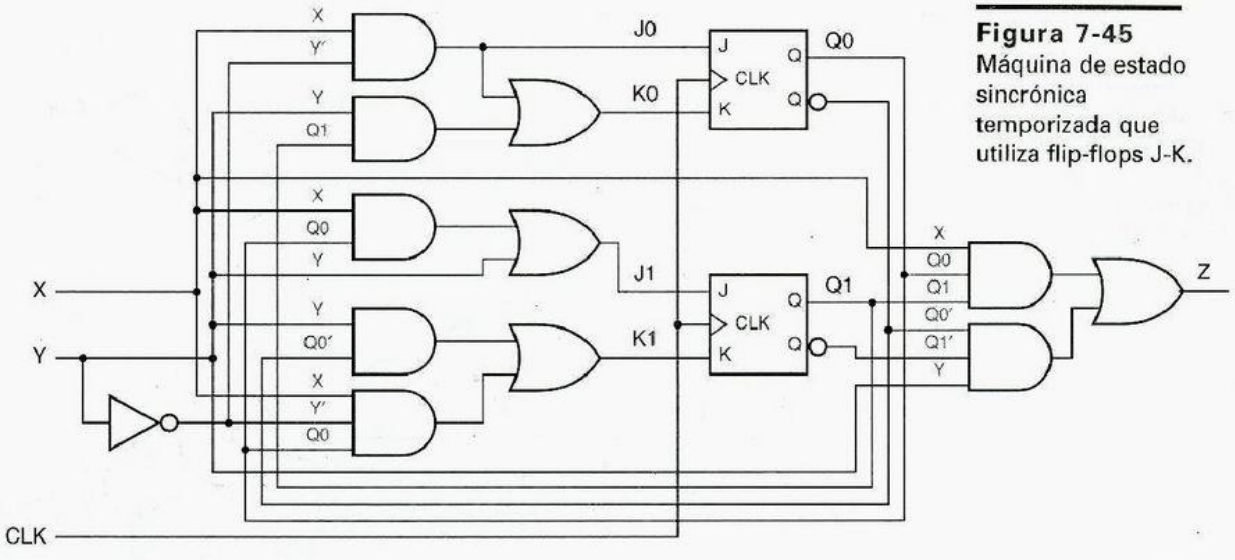


Figura 7-45
Máquina de estado sincrónica temporizada que utiliza flip-flops J-K.

Tabla 7-5
Tablas de transición/salida y estado/salida para la máquina de estado de la figura 7-45.

(a)

	XY			
Q1 Q0	00	01	10	11
00	00, 0	10, 1	01, 0	10, 1
01	01, 0	11, 0	10, 0	11, 0
10	10, 0	00, 0	11, 0	00, 0
11	11, 0	10, 0	00, 1	10, 1
	Q1* Q0*, Z			

(b)

	XY			
S	00	01	10	11
A	A, 0	C, 1	B, 0	C, 1
B	B, 0	D, 0	C, 0	D, 0
C	C, 0	A, 0	D, 0	A, 0
D	D, 0	C, 0	A, 1	C, 1
	S	*, Z		

Sustituyendo en la ecuación característica para flip-flops J-K, obtenemos las ecuaciones de transición:

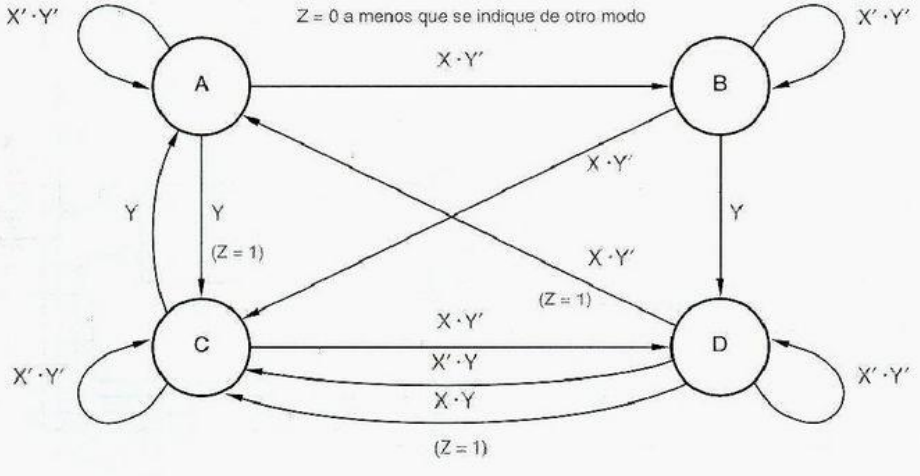
$$Q0* = J0 \cdot Q0' + K0' \cdot Q0$$
$$= X \cdot Y' \cdot Q0' + (X \cdot Y' + Y \cdot Q1)' \cdot Q0$$
$$= X \cdot Y' \cdot Q0' + X' \cdot Y' \cdot Q0 + X' \cdot Q1' \cdot Q0 + Y \cdot Q1' \cdot Q0$$
$$Q1* = J1 \cdot Q1' + K1' \cdot Q1$$
$$= (X \cdot Q0 + Y) \cdot Q1' + (Y \cdot Q0' + X \cdot Y' \cdot Q0)' \cdot Q1$$
$$= X \cdot Q1' \cdot Q0 + Y \cdot Q1' + X' \cdot Y' \cdot Q1 + Y' \cdot Q1 \cdot Q0' + X' \cdot Q1 \cdot Q0 + Y \cdot Q1 \cdot Q0$$

Una tabla de transición basada en estas ecuaciones se muestra en la tabla 7-5(a). Leyendo el diagrama lógico, podemos escribir la ecuación de salida:

$$Z = X \cdot Q1 \cdot Q0 + Y \cdot Q1' \cdot Q0'$$

Los valores de salida resultantes se muestran en cada columna de (a) junto con el estado siguiente. Asignando los nombres de estado A-D, obtenemos la tabla de estado/salida mostrada en (b). Un diagrama de estado correspondiente que emplea expresiones de transición se muestra en la figura 7-46.

Figura 7-46
Diagrama de estado correspondiente a la máquina de estado de la tabla 7-5.



7.4 Diseño de máquina de estado sincrónica temporizada

Los pasos para el diseño de una máquina de estado sincrónica temporizada, comenzando a partir de una especificación o descripción en palabras, son justamente el inverso de los pasos de análisis que utilizamos en la sección anterior:

1. Construir una tabla de estado/salida que corresponde a la especificación o descripción en palabras, empleando nombres de mnemónicos para los estados. (También es posible comenzar con un diagrama de estado; este método se discute en la sección 7.5.) *tabla de estado/salida*
2. (Opcional) Minimizar el número de estados en la tabla de estado/salida. *minimización de estado*
3. Elegir un conjunto de variables de estado y asignar combinaciones de variables de estado a los estados nombrados. *asignación de estado*
4. Sustituir las combinaciones de variables de estado en la tabla de estado/salida para crear una tabla de transición/salida que muestre la siguiente combinación variable-estado deseada y salida para cada combinación de estado/entrada. *tabla de transición/salida*
5. Elegir un tipo de flip-flop (por ejemplo, D o J-K) para la memoria de estado. En la mayoría de los casos, usted ya tendrá una selección en mente al principio del diseño, pero este paso es su última oportunidad de cambiar de idea.
6. Construir una tabla de excitación que muestre los valores de excitación requeridos para obtener el siguiente estado deseado para cada combinación de estado/entrada. *tabla de excitación*
7. Obtener las ecuaciones de excitación a partir de la tabla de excitación. *ecuaciones de excitación*
8. Obtener las ecuaciones de salida a partir de la tabla de transición/salida. *ecuaciones de salida*
9. Dibujar un diagrama lógico que muestre los elementos de almacenamiento de variables de estado realizar las ecuaciones requeridas de excitación y salida. (O realizar las ecuaciones directamente en un dispositivo de lógica programable.) *diagrama lógico*

En esta sección describiremos cada uno de los pasos básicos en el diseño de la máquina de estado. El paso 1 es el más importante, ya que es aquí donde el diseñador realmente *diseña*, pasando a través del proceso creativo de traducir una (quizás ambigua) descripción en lenguaje común de la máquina de estado a una descripción tabular formal. El paso 2 casi nunca es realizado por los diseñadores digitales experimentados, pero los diseñadores dan mucho de su experiencia para sacar adelante el paso 3. *diseño*

Una vez que los tres primeros pasos se han terminado, la totalidad de los pasos restantes se pueden terminar "poniéndose a trabajar", es decir, siguiendo un procedimiento de síntesis bien definido. Los pasos 4 y 6 al 9 son los más tediosos, pero son fácilmente automatizados. Por ejemplo, cuando se diseña una máquina de estado que se realizará en un dispositivo de lógica programable, se puede utilizar un compilador ABEL para hacer el trabajo pesado, como se muestra en la sección 7.11.2. Aún así, es importante que usted comprenda los detalles del procedimiento de síntesis, tanto para darle a usted una apreciación de la función del compilador como para proporcionarle una oportunidad de averiguar lo que realmente está ocurriendo cuando el compilador produce resultados inesperados. Por lo tanto, todos los nueve pasos del procedimiento de diseño de máquina de estado se discuten en el resto de esta sección.

DISEÑO DE TABLA DE ESTADO COMO UNA ESPECIE DE PROGRAMACIÓN

El diseño de una tabla (o de manera equivalente, un diagrama de estado) es un proceso creativo que equivale en muchas formas a escribir un programa de computadora:

- Se puede comenzar con una descripción bastante precisa de entradas y salidas, pero con una descripción posiblemente ambigua de las relaciones deseadas entre ellas, y por lo regular sin pistas acerca de cómo obtener realmente las salidas deseadas a partir de las entradas.
- Durante el diseño se puede tener que identificar y elegir entre diferentes maneras de hacer las cosas, a veces mediante el sentido común, en ocasiones de manera arbitraria.
- Se puede tener que identificar y manejar casos especiales que no estaban incluidos en la descripción original.
- Probablemente se tendrá que hacer un seguimiento de diversas ideas en la mente durante el proceso de diseño.
- Puesto que el proceso de diseño no es un algoritmo, no hay garantía de que se puede completar la tabla de estado o programa utilizando un número finito de estados o líneas de código. Sin embargo, a menos que usted trabaje para el gobierno, debe intentar hacerlo así.
- Cuando finalmente ejecute el programa o máquina de estado, hará exactamente lo que usted haya querido que hiciera, ni menos, ni más.
- Nada nos garantiza que el asunto funcione a la primera vez; seguramente hay que depurar e iterar en todo el proceso.

Aunque el diseño de tabla de estado en un desafío, no hay que sentirse intimidado. Si usted ha hecho esto hasta ahora en su educación, entonces probablemente habrá escrito algunos programas que funcionaron, y puede llegar a ser tan bueno para diseñar tablas de estado.

7.4.1 Ejemplo de diseño de tabla de estado

Existen diversas formas diferentes para describir una tabla de estado de la máquina de estado. Posteriormente, veremos cómo ABEL y VHDL pueden especificar indirectamente tablas de estado. Sin embargo, en esta sección, trataremos solamente con tablas de estado que están especificadas directamente, en el mismo formato tabular que utilizamos en la sección anterior para análisis.

Presentaremos el proceso de diseño de la tabla de estado, además del procedimiento de síntesis en subsecciones posteriores, empleando el problema de diseño simple que se describe a continuación:

> Diseñe una máquina de estado sincrónica temporizada con dos entradas, A y B, y una sola salida Z que es 1 si:
>
> – A tiene el mismo valor en cada uno de los dos anteriores tics de reloj, *o*
>
> – B ha sido 1 desde la última vez que la primera condición fue verdadera.
>
> De otro modo, la salida debería ser 0.

Si el significado de esta especificación no es claro como el cristal para usted en este punto, no se preocupe. Parte de su trabajo como diseñador es convertir una especificación de este tipo en una tabla de estado que no sea ambigua en absoluto; incluso si no coincide con lo que originalmente estaba previsto, cuando menos forma una base para discusión y refinamiento adicionales.

REALIZACIÓN DE UN RESTABLECIMIENTO (RESET) CONFIABLE

Para un funcionamiento adecuado del sistema, el diseño de hardware de una máquina de estado debería asegurar que introduce un estado inicial conocido durante el arranque, tal como el estado INIT en nuestro ejemplo de diseño. La mayoría de los sistemas tienen una señal RESET que es afirmada durante el arranque.

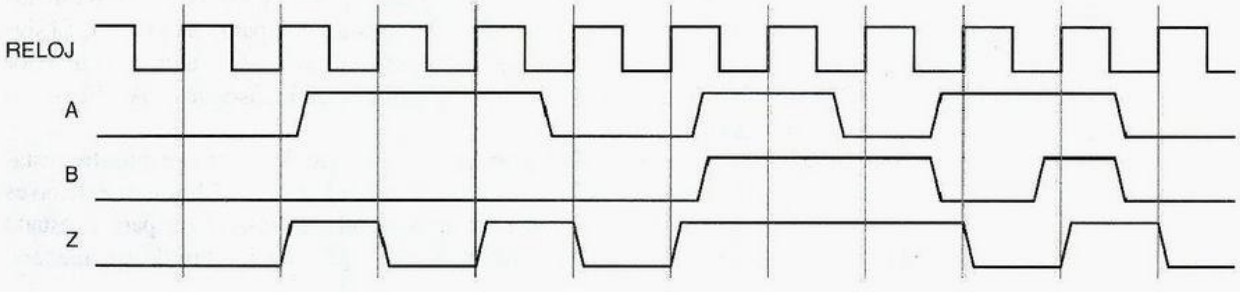
La señal RESET es generada habitualmente por medio de un circuito analógico. Un circuito de restablecimiento de esta clase por lo regular detecta un voltaje (digamos, 4.5 V) cercano al voltaje total de la fuente de alimentación, y sigue con un retardo (digamos, de 100 ms) para asegurar que todos los componentes (incluyendo osciladores) hayan tenido tiempo de estabilizarse antes de que "deje de restablecer" el sistema. El TL7705 de Texas Instruments es un CI de restablecimiento analógico de esta naturaleza; tiene una referencia interna de 4.5 V para el detector y utiliza un capacitor y resistor externos para determinar la constante de tiempo para "dejar de restablecer".

Si para construir una máquina de estado se utilizan flip-flops discretos con entradas de borrado y prestablecimiento asincrónicas, la señal RESET puede aplicarse a estas entradas para obligar a la máquina al estado inicial deseado. Si las entradas de prestablecimiento y de borrado no se encuentran disponibles, o si el restablecimiento debe ser sincrónico (como en sistemas que utilizan microprocesadores de alta velocidad), entonces la señal de RESET puede ser utilizada como otra entrada a la máquina de estado, con todas las entradas del estado siguiente dirigiéndose al estado inicial deseado cuando RESET es afirmada.

Como una "sugerencia" o requerimiento adicional, los problemas de diseño de tabla de estado con frecuencia incluyen diagramas de tiempo que muestran el comportamiento esperado de la máquina de estado para una o más secuencias de entradas. Un diagrama de tiempo de esa clase es poco probable que especifique sin ambigüedad el comportamiento de la máquina para todas las secuencias posibles de entradas, pero, de nueva cuenta, es un buen punto de partida para la discusión y un punto de referencia para verificar los diseños propuestos. La figura 7-47 es un diagrama de tiempo de este tipo para nuestro problema de diseño de la tabla de estado de ejemplo.

El primer paso en el diseño de la tabla de estado es construir una plantilla. A partir de la descripción en palabras, sabemos que nuestro ejemplo es una máquina de Moore: su salida depende solamente del estado actual, es decir de lo que ocurrió en los anteriores periodos del reloj. De este modo, como se ilustra en la figura 7-48(a), proporcionamos una columna del estado siguiente para cada posible combinación de entrada y una columna

Figura 7-47 Diagrama de temporización para la máquina de estado de ejemplo.



(a)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT					0
	...					
	...					
	...					
		S*				

(b)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0					0
Obtención de un 1 en A	A1					0
		S*				

(c)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK	OK	A1	A1	0
Obtención de un 1 en A	A1					0
Obtención de dos entradas A iguales	OK					1
		S*				

(d)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK	OK	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK	OK	0
Obtención de dos entradas A iguales	OK					1
		S*				

Figura 7-48 Evolución de una tabla de estado.

simple para los valores de salida. El orden en el cual se escriben las combinaciones de entrada no afecta esta parte del proceso, pero las hemos escrito en el orden de mapa de Karnaugh para simplificar la derivación de ecuaciones de excitación posteriormente. En una máquina de Mealy omitiríamos la columna de salida y escribiríamos los valores de salida junto con los valores del estado siguiente bajo cada combinación de entrada. La columna más a la izquierda es simplemente un recordatorio en lenguaje coloquial del significado de cada estado o la "historia" asociada con él.

estado inicial

La descripción en palabras no es específica acerca de lo que ocurre cuando esta máquina se inicia primero, así que tendremos que improvisar. Supondremos que cuando se aplica la energía por vez primera al sistema, la máquina entra a un *estado inicial*, llamado INIT en este ejemplo. Escribimos el nombre del estado inicial (INIT) en el primer renglón y dejamos espacio para suficientes renglones (estados) para completar el diseño. También podemos sustituir el valor de Z al estado INIT; el sentido común dice que debería ser cero porque *no* hubo entradas con antelación.

A continuación, debemos sustituir las entradas del estado siguiente para el renglón INIT. La salida Z no puede ser 1 hasta que hayamos visto por lo menos dos entradas en A, de modo que proporcionaremos dos estados A0 y A1, que "recuerden" el valor de A en el anterior tic de reloj, como se ilustra en la figura 7-48(b). En ambos de estos estados Z es 0, puesto que no hemos satisfecho todavía las condiciones para una salida 1. El significado preciso del estado A0 se obtuvo A = 0 en el tic anterior, A ≠ 0 en el tic anterior a ése, y B ≠ 1 en algún instante desde el par anterior de entradas iguales A". El estado A1 se define de manera similar.

En este punto sabemos que nuestra máquina de estado tiene por lo menos tres estados, y hemos creado dos renglones más en blanco para rellenarlos. ¡Mmmmh, ésta no es una tendencia tan buena! A fin de rellenar las entradas del estado siguiente para *un* estado (INIT), tuvimos que crear *dos* nuevos estados A0 y A1. ¡Si continuamos de esta manera,

(a)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK	OK	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK	OK	0
Obtención de dos entradas A iguales	OK	?	OK	OK	?	1
		S*				

(b)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK0	OK0	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK1	OK1	0
Dos A = 0 iguales al final	OK0					1
Dos A = 1 iguales al final	OK1					1
		S*				

(c)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK0	OK0	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK1	OK1	0
Dos A = 0 iguales al final	OK0	OK0	OK0	OK1	A1	1
Dos A = 1 iguales al final	OK1					1
		S*				

(d)

Significado	S	A B 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK0	OK0	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK1	OK1	0
Dos A = 0 iguales al final	OK0	OK0	OK0	OK1	A1	1
Dos A = 1 iguales al final	OK1	A0	OK0	OK1	OK1	1
		S*				

Figura 7-49 Continuación de la evolución de una tabla de estado.

podríamos finalizar hasta con 65,535 estados a la hora de dormir! En vez de ello, deberíamos estar al acecho de estados existentes que tuvieran el mismo significado que los nuevos que podemos crear aparte de eso. Veamos cómo va esto.

En el estado A0, sabemos que la entrada A era 0 en el anterior tic de reloj. Por lo tanto, si A es 0 de nuevo, vamos a un nuevo estado OK con Z = 1, como se muestra en la figura 7-48(c). Si A es 1, entonces no tenemos dos entradas iguales seguidas de modo que vamos al estado A1 para recordar que sólo tenemos un 1. Igualmente, en el estado A1, mostrado en (d), vamos a OK si obtenemos una segunda entrada 1 seguida, o a A0 si obtenemos un 0.

Una vez que llegamos al estado OK, la descripción de la máquina nos dice que podemos permanecer allí mientras B = 1, independientemente de la entrada A, como se muestra en la figura 7-49(a). Si B = 0 tenemos que buscar dos unos o dos ceros seguidos en A de nuevo. Sin embargo, tenemos un pequeño problema en este caso. La entrada A actual puede o no puede ser la segunda entrada igual sucesiva, de manera que todavía podemos estar "OK" o podemos tener que regresar a A0 o A1. Definimos el estado OK en términos demasiado generales: no "recuerda" lo suficiente para decirnos cuál camino tomar.

El problema se resuelve en la figura 7-49(b) al dividir OK en dos estados, OK0 y OK1, que "recuerdan" la entrada anterior A. Todos los estados siguientes para OK0 y OK1 pueden seleccionarse de los estados existentes, como se muestra en (c) y (d). Por ejemplo, si tenemos A = 0 en OK0, sólo podemos permanecer en OK0; no tenemos que crear un nuevo estado que "recuerde" tres ceros seguidos porque la descripción de la máquina no nos exige distinguir ese caso. De este modo, hemos logrado el "cierre" de la tabla de estado, la cual describe ahora una máquina de estado *finito*. Como una sensata verificación, la figura 7-50 repite el diagrama de tiempo de la figura 7-47, numerando los estados que deberían visitarse de acuerdo nuestra tabla de estado final.

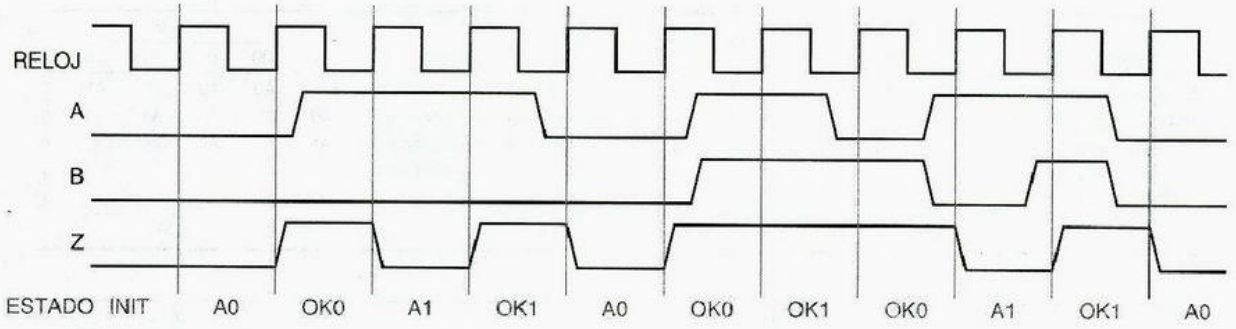


Figura 7-50 Diagrama de temporización y secuencia de estados para la máquina de estado de ejemplo.

7.4.2 Minimización de estado

La figura 7-49(d) es una tabla de estado "mínima" para nuestra descripción en palabras original, en el sentido que contiene el menor número de estados posibles. Sin embargo, la figura 7-51 muestra otras tablas de estado, con más estados, que también hacen el trabajo. Se pueden utilizar procedimientos formales para minimizar el número de estados en esas tablas.

estados equivalentes

La idea básica de los procedimientos de minimización formal es identificar los *estados equivalentes*, donde dos estados son equivalentes si es imposible distinguirlos al observar solamente las *salidas* actual y futura de la máquina (y *no* las variables de estado internas). Se pueden reemplazar un par de estados equivalentes por un solo estado.

Dos estados S1 y S2 son equivalentes si dos condiciones son verdaderas. Primero, S1 y S2 deben producir los mismos valores a la salida(s) de la máquina de estado; en una máquina de Mealy, esto debe ser verdadero para todas las combinaciones de entrada. En segundo lugar, para cada combinación de entrada, S1 y S2 deben tener ya sea el mismo estado siguiente o estados siguientes equivalentes.

De esta manera, un procedimiento de minimización de estado formal muestra que los estados OK00 y OKA0 en la figura 7-51(a) son equivalentes porque producen la misma salida y sus entradas del estado siguiente son idénticas. Puesto que los estados son equivalentes, el estado OK00 puede ser eliminado y sus incidencias en la tabla

Figura 7-51 Tablas de estado equivalentes no mínimas para la figura 7-49(d).

(a)

Significado	S	AB 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK00	OK00	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK11	OK11	0
Obtención de un 00 en A	OK00	OK00	OK00	OKA1	A1	1
Obtención de un 11 en A	OK11	A0	OKA0	OK11	OK11	1
OK, obtención de un 0 en A	OKA0	OK00	OK00	OKA1	A1	1
OK, obtención de un 1 en A	OKA1	A0	OKA0	OK11	OK11	1
		S*				

(b)

Significado	S	AB 00	01	11	10	Z
Estado inicial	INIT	A0	A0	A1	A1	0
Obtención de un 0 en A	A0	OK00	OK00	A1	A1	0
Obtención de un 1 en A	A1	A0	A0	OK11	OK11	0
Obtención de un 00 en A	OK00	OK00	OK00	A001	A1	1
Obtención de un 11 en A	OK11	A0	A110	OK11	OK11	1
Obtención de un 001 en A, B = 1	A001	A0	AE10	OK11	OK11	1
Obtención de un 110 en A, B = 1	A110	OK00	OK00	AE01	A1	1
Obtención de bb... 10 en A, B = 1	AE10	OK00	OK00	AE01	A1	1
Obtención de bb... 01 en A, B = 1	AE01	A0	AE10	OK11	OK11	1
		S*				

reemplazadas por OKA0, o viceversa. Del mismo modo, los estados OK11 y OKA1 son equivalentes.

Para minimizar la tabla de estado en la figura 7-51(b), un procedimiento formal debe emplear un poco de razonamiento circular. Los estados OK00, A110 y AE10 producen la misma salida y tienen entradas de estado siguiente casi iguales, de modo que pueden ser equivalentes. Son equivalentes solamente si A001 y AE01 son equivalentes. De manera similar, OK11, A001 y AE01 son equivalentes sólo si A110 y AE10 son equivalentes. En otras palabras, los estados en el primer conjunto son equivalentes si los estados en el segundo conjunto lo son, y viceversa. Así, sólo procedamos y digamos que son equivalentes.

¿ES TODO ESTO REALMENTE NECESARIO?

Los detalles de los procedimientos formales de minimización de estado se discuten en libros de texto avanzados, citados en las Referencias. Sin embargo, estos procedimientos rara vez son empleados por la mayoría de los diseñadores digitales. Al hacer coincidir cuidadosamente los significados de estado con los requerimientos del problema, los diseñadores digitales con experiencia producen tablas de estado para problemas pequeños con un mínimo o aproximadamente un mínimo de número de estados, sin utilizar un procedimiento de minimización formal. También, existen situaciones donde al *incrementar* el número de estados se puede simplificar el diseño o reducir su costo, de manera que inclusive un procedimiento de minimización de estado automatizado no necesariamente ayuda. Un diseñador puede hacer más que simplificar una máquina de estado durante la fase de asignación de estado del diseño, discutido en la subsección siguiente.

7.4.3 Asignación de estado

El siguiente paso en el proceso de diseño es determinar cuántas variables binarias se requieren para representar los estados en la tabla de estado, y asignar una combinación específica a cada estado nombrado. Podemos llamar a la combinación binaria asignada a un estado particular un *estado codificado*. El *número total de estados* en una máquina con n flip-flops es 2^n, de modo que el número de flip-flops necesarios para codificar s estados es $\lceil \log_2 s \rceil$, el entero más pequeño mayor o igual que $\log_2 s$.

estado codificado
número total de estados

Para referencia, la tabla de estado/salida de nuestra máquina de ejemplo se repite en la tabla 7-6. Tiene cinco estados, de modo que requiere tres flip-flops. Por supuesto, tres flip-flops proporcionan un total de ocho estados, por lo tanto quedan sin utilizar $8 - 5 = 3$ *estados*. Discutiremos alternativas para manejar los estados sin utilizar al final de esta subsección. Justo ahora, tenemos que ocuparnos de muchas opciones para los cinco estados codificados.

estados sin utilizar

ESTADO INICIAL VERSUS INACTIVO

La máquina de estado ejemplo de esta subsección visita su estado inicial solamente durante el restablecimiento (reset). Muchas máquinas son diseñadas en cambio con un estado "inactivo" "idle" al cual se entra tanto en el restablecimiento como cuando la máquina no tiene nada en particular qué hacer.

Tabla 7-6
Tabla de estado y salida para el problema ejemplo.

	A B				
S	***00***	***01***	***11***	***10***	***Z***
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
			S*		

La asignación más simple de s estados codificados a 2^n estados posibles es emplear los primeros s enteros binarios en orden de conteo binario, como se muestra en la primera columna de asignación de la tabla 7-7. Sin embargo, la asignación de estado más simple no siempre conduce a las más simples ecuaciones de excitación, ecuaciones de salida y circuito lógico resultante. De hecho, la asignación de estado con frecuencia tiene un efecto mayor sobre el costo del circuito, y puede interactuar con otros factores, tales como la selección de los elementos de almacenamiento (por ejemplo, flip-flops D vs. J-K) y el enfoque de realización para la lógica de salida y excitación (por ejemplo, un producto de sumas o un diseño ad hoc).

De esta forma, ¿cómo elegimos la mejor asignación de estado para un problema dado? En general, la única manera formal para encontrar la *mejor* asignación es probar con *todas* las asignaciones. Esto es demasiado trabajo, incluso para estudiantes. En su lugar, la mayoría de los diseñadores digitales confían en su experiencia y diversas pautas prácticas para efectuar razonables asignaciones de estado:

- Elija un estado codificado inicial en el cual la máquina fácilmente pueda ser forzada a reinicializar (00... 00 u 11... 11 en circuitos típicos).
- Minimice el número de variables de estado que cambien en cada transición.

PRECAUCIÓN: MATEMÁTICAS

El número de maneras diferentes para elegir m estados codificados de un conjunto de n estados posibles está dado por un *coeficiente binomial*, denotado por $\binom{n}{m}$, cuyo valor es $\frac{n!}{m! \cdot (n-m)!}$. (Hicimos uso de coeficientes binomiales anteriormente en la sección 2.10 en el contexto de la codificación decimal.) En nuestro ejemplo, existen $\binom{8}{5}$ maneras diferentes de elegir cinco estados codificados de ocho estados posibles, y 5! formas de asignar los cinco estados nombrados para cada elección diferente. De modo que tenemos $\frac{8!}{5! \cdot 3!} \cdot 5!$ o 6,720 formas distintas de asignar los cinco estados de nuestra máquina de ejemplo, para combinaciones de tres variables de estado binarias. No vamos a tener tiempo de examinar la totalidad de ellas.

Tabla 7-7
Posibles asignaciones de estado para la máquina de estado en la tabla 7-6.

Nombre de estado	Asignación			
	Q1–Q3 más simple	*Q1–Q3 descompuesta*	*Q1–Q5 estado activo uno*	*Q1–Q4 casi estado activo uno*
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

- Maximice el número de variables de estado que no cambian en un grupo de estados relacionados (es decir, un grupo de estados en el cual la mayoría de las transiciones permanezcan en el grupo).
- Explote las simetrías en la especificación del problema y las simetrías correspondientes en la tabla de estado. Es decir, suponga que un estado o grupo de estados significa casi lo mismo que otro. Una vez que se haya establecido una asignación para el primero, una asignación similar, que difiera solamente en un bit, debería utilizarse para el segundo.
- Si existen estados sin utilizar (es decir, si $s < 2^n$ donde $n = \lceil \log_2 s \rceil$), entonces elija la "mejor" de las combinaciones disponibles de variable de estado para conseguir las metas anteriores. Esto es, no limite la elección de estados codificados a los primeros s enteros de n bits.
- Descomponga el conjunto de variables de estado en bits o campos individuales, donde cada bit o campo tiene un significado bien definido con respecto a los efectos de entrada o comportamiento de salida de la máquina.
- Considere utilizar más del número mínimo de variables de estado para hacer una posible asignación descompuesta.

Algunas de estas ideas se encuentran incorporadas en la asignación de estado "descompuesta" en la tabla 7-7. Como antes, el estado inicial es 000, que es fácil de forzar ya sea asincrónicamente (aplicando la señal RESET a las entradas CLR del flip-flop) o sincrónicamente (al operar con AND a RESET' con todas las entradas del flip-flop D). Después de este punto, la asignación toma ventaja del hecho que existen solamente cuatro estados además de INIT, que es un estado bastante "especial" que no se vuelve a cargar después que arranca la máquina. Por lo tanto, Q1 puede emplearse para indicar si la máquina se encuentra o no en el estado INIT, y Q2 y Q3 se emplean para distinguir entre los cuatro estados no INIT.

Los estados no INIT en la columna "descompuesta" de la tabla 7-7 parecen haber sido asignados en orden de conteo binario, pero esto sólo es una coincidencia. Los bits de estado Q2 y Q3 en realidad tienen significados individuales en el contexto de las

entradas y salida de la máquina de estado. Q3 proporciona el valor anterior de A, y Q2 indica que las condiciones para una salida 1 se satisfacen en el estado actual. Al descomponer los significados del bit de estado de esta manera, podemos esperar que la lógica de salida y estado siguiente sean más simples que en una asignación "aleatoria" de combinaciones Q2, Q3 a los estados no INIT. Continuaremos el diseño de la máquina de estado basado en esta asignación en subsecciones posteriores.

asignación de estado activo uno (one-hot)

Otra asignación útil de estado, que puede adaptarse a cualquier máquina de estado, es la *asignación de estado activo uno* (one-hot) mostrada en la tabla 7-7. Esta asignación utiliza más del número mínimo de variables de estado: emplea un bit por estado. Además de ser simple, una asignación de estado activo uno tiene la ventaja de conducir por lo regular a ecuaciones de excitación pequeñas, puesto que cada flip-flop debe ser establecido a 1 para las transiciones en solamente un estado. Una desventaja obvia de una asignación de estado activo, uno, en especial para máquinas con muchos estados, es que requiere (mucho) más que el número mínimo de flip-flops. Sin embargo, la codificación de estado activo uno es ideal para una máquina con s estados que se requiere para tener un conjunto de salidas codificadas 1 de s indicando su estado actual. Las salidas flip-flop codificadas en estado activo uno, se pueden utilizar directamente para este propósito, sin lógica de salida combinacional adicional.

La última columna de la tabla 7-7 es una "casi asignación de estado activo uno" que emplea la combinación de "no estado activo" para el estado inicial. Esto tiene mucho sentido por dos razones: es fácil inicializar la mayoría de los dispositivos de almacenamiento al estado de ceros en su totalidad y el estado inicial en esta máquina nunca es vuelto a visitar una vez que la máquina arranca. La terminación del diseño de la máquina de estado que utiliza esta asignación de estado se considera en los ejercicios 7.37 y 7.40.

estados sin utilizar

Prometimos al principio considerar la disposición de los *estados sin utilizar* cuando el número de estados disponible con n flip-flops, 2^n, fuera mayor que el número de estados requeridos, s. Existen dos enfoques que tienen sentido, dependiendo de los requerimientos de la aplicación:

- *Riesgo mínimo.* Este enfoque supone que es posible para la máquina de estado de algún modo llegar a uno de los estados sin utilizar (o "ilegales"), quizás a consecuencia de una falla del hardware, una entrada inesperada o un error de diseño. Por lo tanto, todas las combinaciones de variable de estado sin utilizar son identificadas y se hacen entradas de estado siguiente explícitas de modo que, para cualquier combinación de entrada, los estados sin utilizar se vayan al estado "inicial", al estado "libre" o algún otro estado "seguro". Ésta es una consecuencia automática de algunas metodologías de diseño si el estado inicial es codificado 00 . . . 00.
- *Costo mínimo.* Este enfoque supone que la máquina nunca entrará en un estado sin utilizar. Por lo tanto, en las tablas de transición y excitación, las entradas del estado siguiente de los estados sin utilizar se pueden identificar como "sin importancia". En la mayoría de los casos esto simplifica la lógica de excitación. Sin embargo, si es que la máquina entra a algún estado, su comportamiento puede ser inestable.

Examinaremos ambos enfoques a medida que completamos el diseño de nuestra máquina de estado de ejemplo.

7.4.4 Síntesis utilizando flip-flops D

Una vez que hemos asignado estados codificados a los estados nombrados de una máquina, el resto del proceso de diseño consiste en "escribir mucho". De hecho, en la sección 7.11.2 describiremos las herramientas de software que pueden hacer el trabajo por usted. Sin embargo, sólo para que usted llegue a apreciar esas herramientas, llevaremos a cabo el proceso a mano en esta subsección.

Los estados nombrados son sustituidos por estados codificados en la (posiblemente minimizada) tabla de estado para obtener una *tabla de transición*. La tabla de transición muestra el siguiente estado codificado para cada combinación de entrada y estado codificado actual. La tabla 7-8 muestra la tabla de salida y transición que se obtiene de la máquina de estado, del ejemplo de la tabla 7-6, que utiliza la asignación "descompuesta" de la tabla 7-7.

tabla de transición

El paso siguiente es escribir una *tabla de excitación* que muestre, para cada combinación de entrada y estado codificado, los valores de entrada de excitación de flip-flop necesarios para hacer que la máquina vaya al siguiente estado codificado deseado. La estructura y contenido de esta tabla depende del tipo de flip-flops que se utilicen (D, J-K, T, etc.). *Por lo regular* tenemos un tipo de flip-flop particular en mente al comienzo de un diseño, y *desde luego* que lo haremos en esta subsección, dado su título. De hecho, la mayor parte de los diseños de máquinas de estado en la actualidad hacen uso de flip-flops D, debido a su disponibilidad tanto en encapsulados discretos como en dispositivos lógicos programables, y a consecuencia de su facilidad de uso (compárese con los flip-flops J-K en la siguiente subsección).

tabla de excitación

De todos los tipos de flip-flops, un flip-flop D tiene la ecuación característica más simple, Q* = D. Cada flip-flop D en una máquina de estado tiene una entrada de excitación simple, D, y la tabla de excitación debe mostrar el valor requerido para cada entrada D del flip-flop para cada combinación estado-codificado/salida. La tabla 7-9 muestra la tabla de excitación para nuestro problema de ejemplo. Puesto que D = Q*, la tabla de excitación es idéntica a la tabla de transición, excepto por las etiquetas de sus entradas. De este modo, con flip-flops D, no se necesita en realidad escribir una tabla de excitación por separado; se puede denominar solamente a la primera tabla como una *tabla de transición/excitación*.

tabla de transición/excitación

La tabla de excitación es como una tabla de verdad para tres funciones lógicas combinacionales (D1, D2, D3) de cinco variables (A, B, Q1, Q2, Q3). En consecuencia,

	AB				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1* Q2* Q3*				

Tabla 7-8
Tabla de transición y salida para el problema de ejemplo.

Tabla 7-9
Tabla de salida y excitación para la tabla 7-8 que utiliza flip-flops D.

	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				

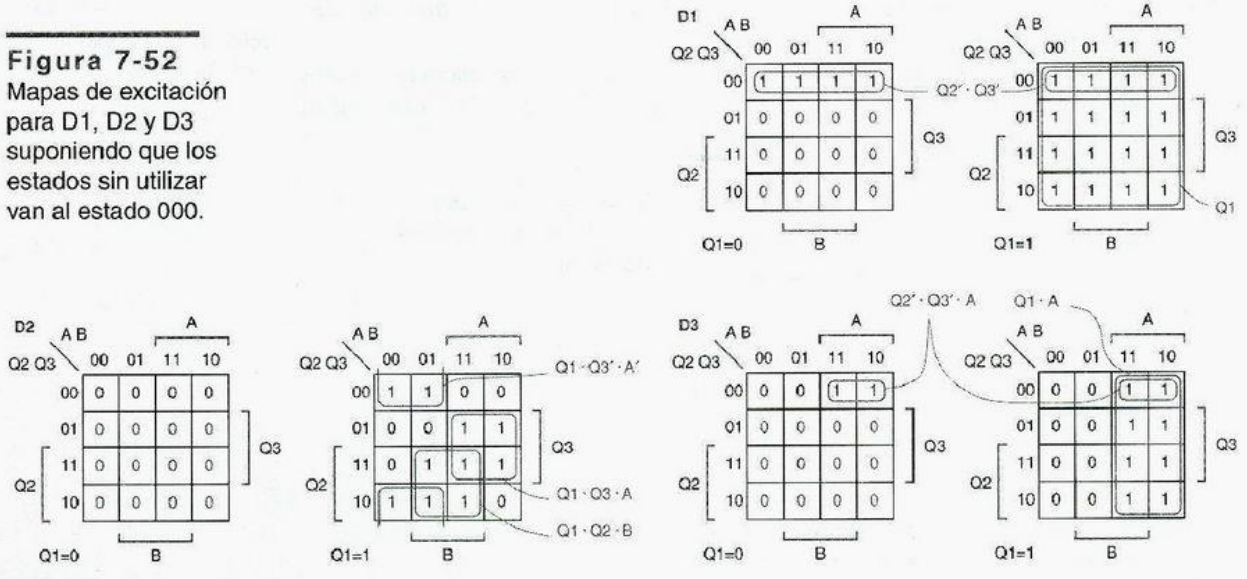
podemos diseñar circuitos para realizar estas funciones utilizando cualquiera de los métodos de diseño combinacional a nuestra disposición. En particular, podemos transferir la información en la tabla de excitación a mapas de Karnaugh, lo que podemos llamar *mapas de excitación*, y hallar una expresión mínima de producto de sumas para cada función.

mapas de excitación

Los mapas de excitación para nuestra máquina de estado de ejemplo se muestran en la figura 7-52. Cada función, tal como D1, tiene cinco variables y por lo tanto utiliza un *mapa de Karnaugh de 5 variables*. Un mapa de 5 variables es dibujado como un par de mapas de 4 variables, donde las celdas en la misma posición en los dos mapas se consideran como adyacentes. Estos mapas son un poco difíciles de manejar, pero si usted quiere diseñar a mano cualquiera de las más triviales máquinas de estado, va a atorarse con los mapas de 5 variables y peores. Por lo menos tuvimos la previsión de etiquetar las combinaciones de entrada de la tabla de estado original en el orden del mapa de Karnaugh, lo que la hace más fácil para transferir información a los mapas en este paso. Sin embargo, advierta que los *estados* no fueron asignados en el orden del mapa de Karnaugh; en particular, los renglones para los estados 110 y 111 se encuentran en el orden opuesto en el mapa que en la tabla de excitación.

mapa de Karnaugh de 5 variables

Figura 7-52
Mapas de excitación para D1, D2 y D3 suponiendo que los estados sin utilizar van al estado 000.



Es en este paso, transfiriendo la tabla de excitación a los mapas de excitación, que descubrimos por qué la tabla de excitación no es del todo una tabla de verdad: no especifica valores funcionales para *todas* las combinaciones de entrada. En particular, la información del estado siguiente para los estados sin utilizar, 001, 010 y 011, no está especificada. Aquí debemos hacer una elección, discutida en la subsección anterior, entre una estrategia de riesgo mínimo y una de costo mínimo para manejar los estados sin utilizar. La figura 7-52 ha tomado el enfoque del riesgo mínimo: el estado siguiente para cada estado sin utilizar y combinación de entrada es 000, el estado INIT. Los tres renglones de ceros en color gris en cada mapa de Karnaugh son el resultado de esta elección. Con los mapas completamente rellenados, podemos ahora obtener expresiones de suma de productos mínimas para las entradas de excitación de los flip-flops:

$$D1 = Q1 + Q2' \cdot Q3'$$
$$D2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A + Q1 \cdot Q2 \cdot B$$
$$D3 = Q1 \cdot A + Q2' \cdot Q3' \cdot A$$

Una ecuación de salida puede desarrollarse fácilmente en forma directa de la información en la tabla 7-9. La ecuación de salida es más simple que las ecuaciones de excitación, porque la salida es una función solamente del estado. Podríamos utilizar un mapa de Karnaugh, pero es más sencillo encontrar una función de salida de riesgo mínimo de manera algebraica, escribiéndola como la suma de los dos estados codificados (110 y 111) en los cuales Z es 1:

$$Z = Q1 \cdot Q2 \cdot Q3' + Q1 \cdot Q2 \cdot Q3$$
$$= Q1 \cdot Q2$$

En este punto, hemos casi concluido el diseño de la máquina de estado. Si la máquina de estado va a construirse con flip-flops y compuertas discretas, entonces el paso final es dibujar un diagrama lógico. Por otra parte, si utilizamos un dispositivo de lógica programable, entonces sólo tenemos que introducir las ecuaciones de excitación y de salida en un archivo de computadora que especifique cómo programar el dispositivo; un ejemplo se muestra en la sección 7.11.1. O, si fuéramos previsores, especificaríamos la máquina empleando un lenguaje de descripción de máquina de estado como ABEL en primer lugar (sección 7.11.2) ¡y la computadora haría todo el trabajo en esta subsección para nosotros!

SOLUCIÓN DE COSTO MÍNIMO

Si elegimos en nuestro ejemplo obtener ecuaciones de excitación de costo mínimo, escribimos elementos "sin importancia" en las entradas de estado siguiente para los estados sin utilizar. Las d en color gris en la figura 7-53 son el resultado de esta selección. Las ecuaciones de excitación obtenidas de este mapa son algo más simples que antes:

$$D1 = 1$$
$$D2 = Q1 \cdot Q3' \cdot A' + Q3 \cdot A + Q2 \cdot B$$
$$D3 = A$$

Para una función de salida de costo mínimo, el valor de Z es un "sin importancia" para los estados sin utilizar. Esto conduce a una función de salida incluso más simple, $Z = Q2$. El diagrama lógico para la solución de costo mínimo se muestra en la figura 7-54.

Figura 7-53
Mapas de excitación para D1, D2 y D3 suponiendo que los estados siguientes de los estados sin utilizar son "sin importancia".

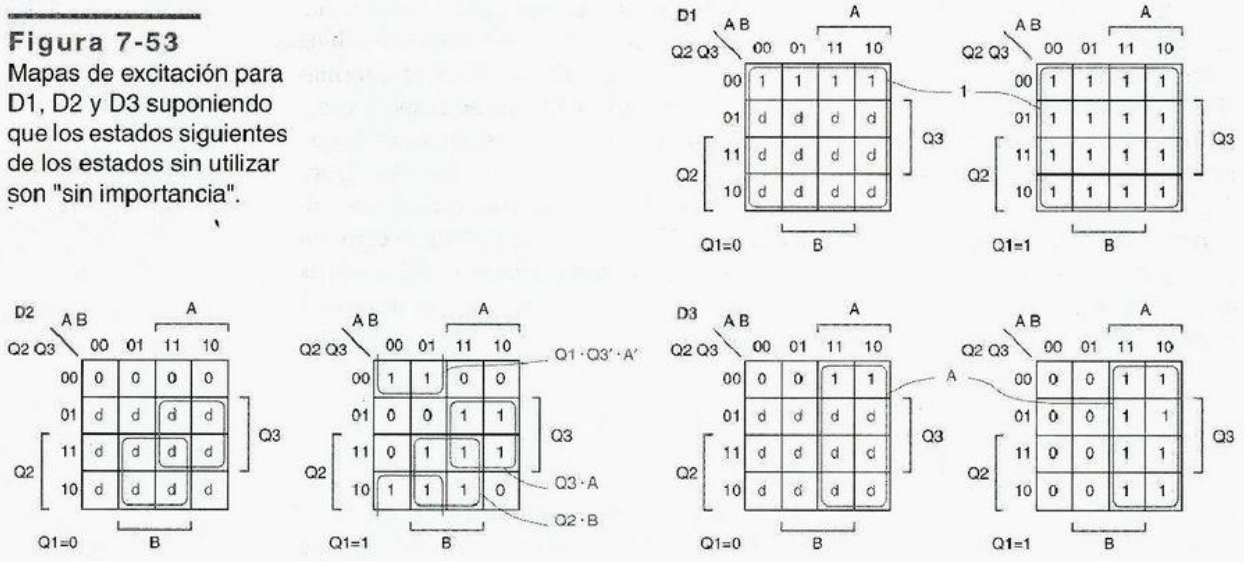
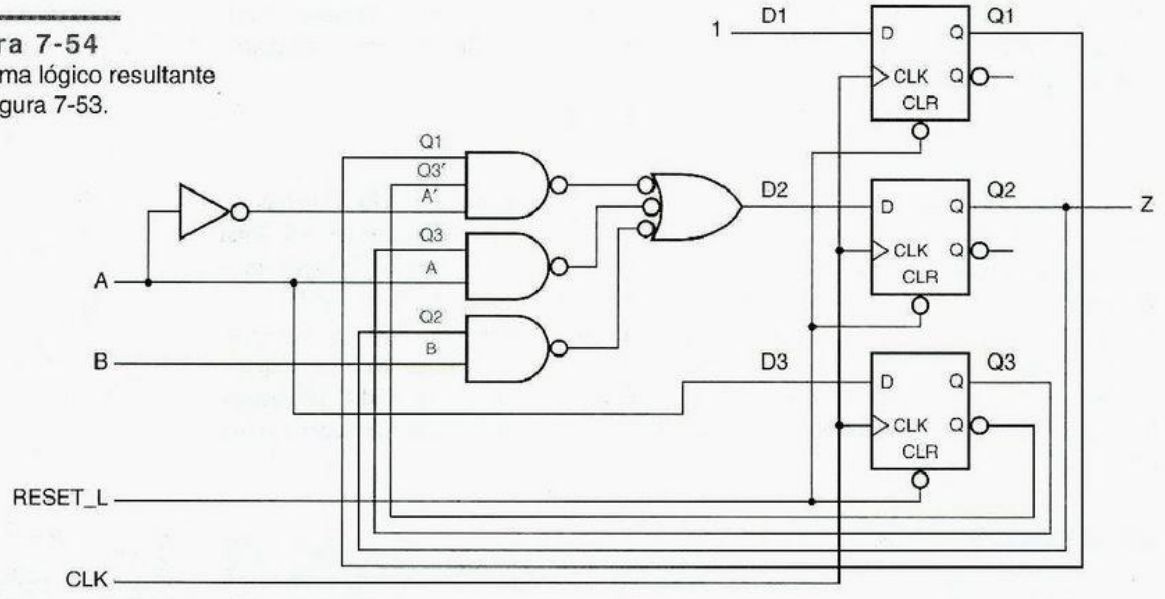


Figura 7-54
Diagrama lógico resultante de la figura 7-53.



*7.4.5 Síntesis utilizando flip-flops J-K

Alguna vez, los flip-flops J-K fueron populares para diseños de máquinas de estado SSI discretas, puesto que un flip-flop J-K contiene más funcionalidad que un flip-flop D en el mismo encapsulado de dimensiones SSI. Por "más funcionalidad" se entiende que la combinación de entradas J y K producen más posibilidades para controlar el flip-flop que una entrada D simple. Como resultado, la lógica de excitación de una máquina de estado puede ser más simple si utilizamos flip-flops J-K en lugar de flip-flops D, lo que reduce el conteo del encapsulado cuando se utilizaron compuertas SSI para la lógica de excitación.

SÓLO POR DIVERSIÓN

Mientras que minimizar la lógica de excitación era un gran lío en los días del diseño basado en SSI, las reglas del juego han cambiado con los PLD y ASIC. Como usted podrá determinar gracias a sus conocimientos de la estructura ADN-OR de los PLDs combinacionales, la necesidad de proporcionar arreglos AND-OR separados para las entradas J y K de un flip-flop J-K sería una desventaja distintiva en un PLD secuencial.

En las tecnologías ASIC, los flip-flops J-K ya no son tan deseables. Por ejemplo, en la serie LCA500K de LSI Logic Corp. de arreglos de compuerta CMOS, una macrocelda de flip-flop D FD1QP utiliza 7 "celdas de compuerta", mientras que una macrocelda de flip-flop J-K FJK1QP utiliza 9 celdas de compuerta, aproximadamente 25% de área adicional del chip. Por lo tanto, un diseño más eficiente en costo se obtiene al aplicar flip-flops D y utilizar el área adicional del chip para la lógica de excitación más compleja, donde es realmente necesario.

Con todo, esta subsección describe el proceso de síntesis J-K "sólo por diversión".

Durante el paso de asignación de estado, el procedimiento de diseño con flip-flops J-K es básicamente el mismo que con flip-flops D. La única diferencia es que un diseño puede seleccionar una asignación de estado ligeramente distinto, conociendo la clase de comportamiento que puede obtenerse fácilmente de los flip-flops J-K (por ejemplo, "conmutando" al establecer J y K a 1).

La gran diferencia se presenta en la obtención de una tabla de excitación a partir de la tabla de transición. Con flip-flops D, las dos tablas son idénticas; si utilizamos la ecuación característica de D, $Q^* = D$, simplemente sustituimos $D = Q^*$ para cada entrada. Con flip-flops J-K, cada entrada de la tabla de excitación tiene dos veces tantos bits como en la tabla de transición, puesto que hay dos entradas de excitación por flip-flop.

La ecuación característica de un flip-flop J-K, $Q^* = J \cdot Q' + K' \cdot Q$, no puede reacomodarse para obtener ecuaciones independientes para J y K. En su lugar, los valores requeridos para J y K se expresan como funciones de Q y Q* en una *tabla de aplicación* J-K, tabla 7-10. De acuerdo al primer renglón, si Q es actualmente 0, todo lo que se requiere para obtener 0 como el siguiente valor de Q es establecer J a 0; el valor de K no importa. De manera semejante, de acuerdo con el tercer renglón, si Q es actualmente 1, el siguiente valor de Q será 0 si K es 1, sin tener en cuenta el valor de J. Cada transición deseada puede obtenerse por cualquiera de dos diferentes combinaciones en las entradas J y K, de modo que obtenemos una entrada "sin importancia" en cada renglón de la tabla de aplicación.

tabla de aplicación J-K

Para obtener una tabla de excitación J-K, el diseñador debe examinar tanto el valor actual como el valor siguiente deseado de cada bit de estado en la tabla de transición y

Q	Q*	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

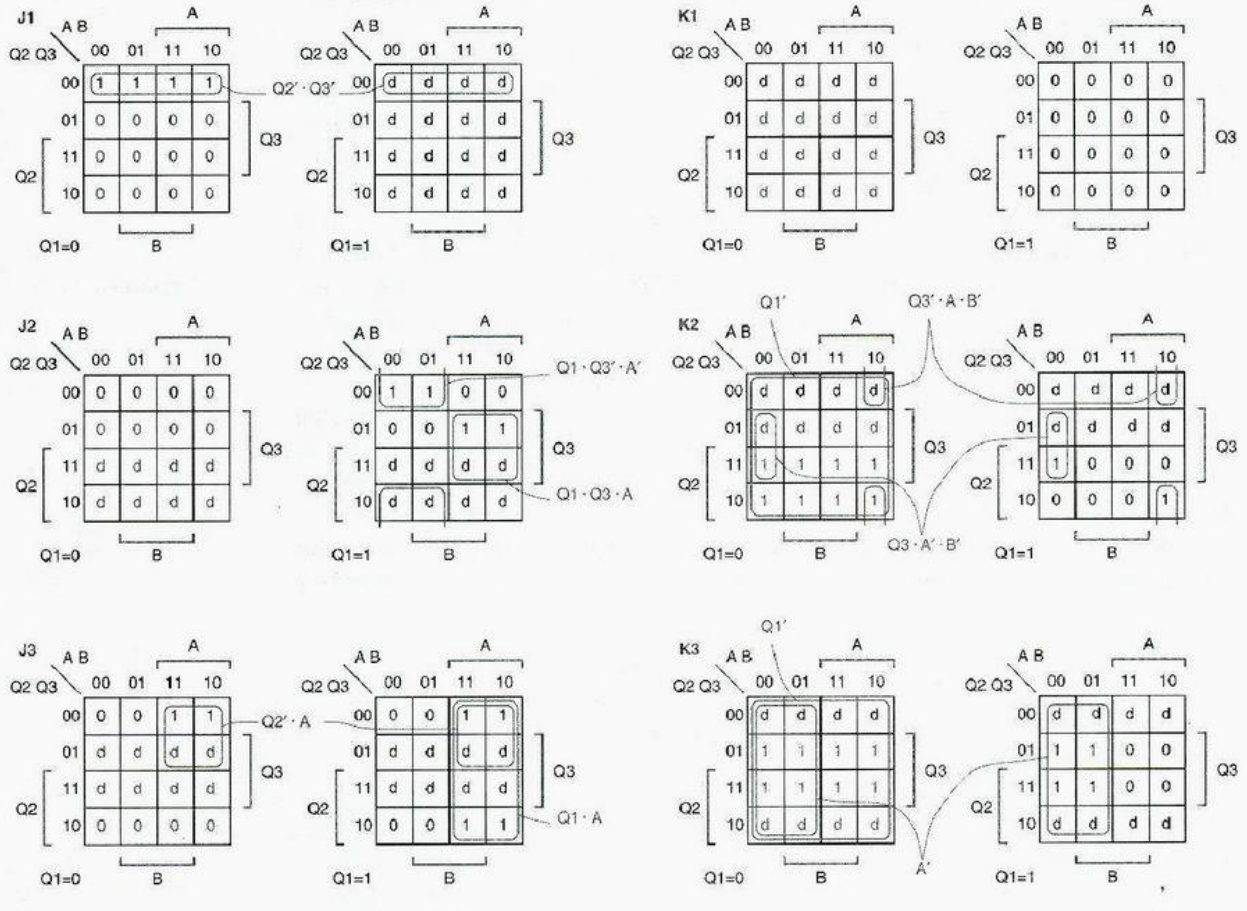
Tabla 7-10
Tabla de aplicación para flip-flops J-K.

sustituir el correspondiente par de valores J y K de la tabla de aplicación. Para la tabla de transición 7-8 , estas sustituciones producen la tabla de excitación en la tabla 7-11. Por ejemplo, en estado 100 bajo la combinación de entrada 00, Q1 es 1 y el Q1* requerido es 1; por lo tanto, se introduce "d0" para J1 K1. Para la misma combinación estado/entrada, Q2 es 0 y Q2* es 1, de modo que se introduce "1d" para J2 K2. Obviamente, se necesita cierta paciencia y cuidado para llenar toda la tabla de excitación (un trabajo que se deja mejor para una computadora).

Como en el ejemplo de síntesis de D en la subsección anterior, la tabla de excitación es *casi* una tabla de verdad para las funciones de excitación. Esta información se transfiere a mapas de Karnaugh en la figura 7-55.

La tabla de excitación no especifica los estados siguientes para los estados sin utilizar, de modo que una vez más debemos elegir entre los enfoques de mínimo riesgo y mínimo costo. Las entradas en color en los mapas de Karnaugh resultan de tomar el enfoque del mínimo riesgo.

Figura 7-55 Mapas de excitación para J1, K1, J2, K2, J3 y K3, suponiendo que los estados sin utilizar se van al estado 000.



	AB				
Q1 Q2 Q3	00	01	11	10	Z
000	1d, 0d, 0d	1d, 0d, 0d	1d, 0d, 1d	1d, 0d, 1d	0
100	d0, 1d, 0d	d0, 1d, 0d	d0, 0d, 1d	d0, 0d, 1d	0
101	d0, 0d, d1	d0, 0d, d1	d0, 1d, d0	d0, 1d, d0	0
110	d0, d0, 0d	d0, d0, 0d	d0, d0, 1d	d0, d1, 1d	1
111	d0, d1, d1	d0, d0, d1	d0, d0, d0	d0, d0, d0	1
	J1 K1, J2 K2, J3 K3				

Tabla 7-11
Tabla de salida y excitación para la máquina de estado de la tabla 7-8, empleando flip-flops J-K.

Note que aun cuando el siguiente estado "seguro" para los estados sin utilizar es 000, no colocamos únicamente ceros en las correspondientes celdas del mapa, como podíamos hacerlo en el caso D. En su lugar, tendríamos que trabajar todavía con la tabla de aplicación para determinar la combinación adecuada de J y K necesaria para obtener $Q^* = 0$ para cada entrada de estado sin utilizar, una vez más un proceso tedioso y propenso a errores.

De los mapas en la figura 7-55, podemos obtener las ecuaciones de excitación de suma de productos:

$$J1 = Q2' \cdot Q3' \qquad K1 = 0$$

$$J2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A \qquad K2 = Q1' + Q3' \cdot A \cdot B' + Q3 \cdot A' \cdot B'$$

$$J3 = Q2' \cdot A + Q1 \cdot A \qquad K3 = Q1' + A'$$

Estas ecuaciones toman dos compuertas adicionales para realizarse en comparación con las ecuaciones de riesgo mínimo de la subsección anterior, que utilizan flip-flops D, de modo que los flip-flops J-K no nos ahorran nada en este ejemplo, menos todo el tiempo de diseño.

SOLUCIÓN DE MÍNIMO COSTO

En el anterior ejemplo de diseño, los mapas de excitación para el enfoque de costo mínimo podrían haber sido algo más fáciles de construir, puesto que podríamos haber puesto simplemente las d en todas las entradas de estado sin utilizar. Las ecuaciones de excitación de suma de productos obtenidas de los mapas de costo mínimo (no mostrados) son como se ve a continuación:

$$J1 = 1 \qquad K1 = 0$$

$$J2 = Q1 \cdot Q3' \cdot A' + Q3 \cdot A \qquad K2 = Q3' \cdot A \cdot B' + Q3 \cdot A' \cdot B'$$

$$J3 = A \qquad K3 = A'$$

La codificación de estado para el circuito J-K es la misma que en el circuito D, de manera que la ecuación de salida viene siendo la misma, $Z = Q1 \cdot Q2$ para riesgo mínimo, $Z = Q2$ para costo mínimo.

Un diagrama lógico correspondiente a las ecuaciones de costo mínimo se muestra en la figura 7-56. Este circuito tiene dos compuertas más que el circuito D de costo mínimo en la figura 7-54, de modo que los flip-flops J-K aún no nos han ahorrado nada.

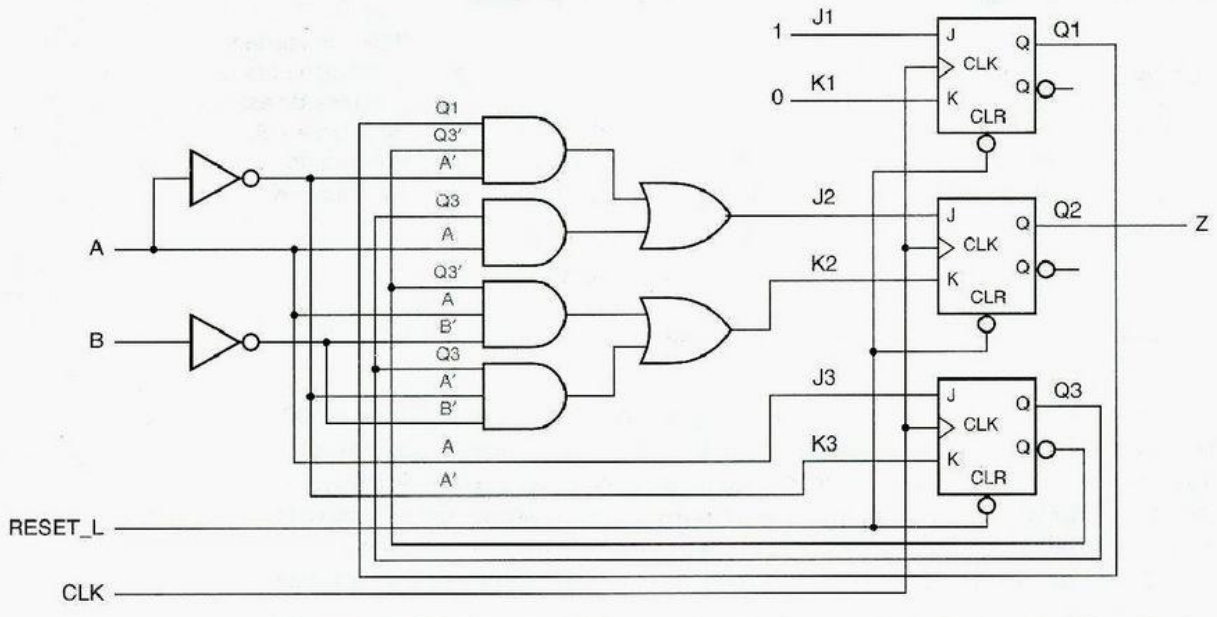


Figura 7-56 Diagrama lógico para la máquina de estado de ejemplo utilizando flip-flops J-K y lógica de excitación de costo mínimo.

7.4.6 Más ejemplos de diseño que utilizan flip-flops D

Concluiremos esta sección con dos ejemplos adicionales de diseño para una máquina de estado que utiliza flip-flops D. El primer ejemplo es una "máquina de conteo de unos":

> Diseñe una máquina de estado sincrónica temporizada con dos entradas X y Y, y una salida, Z. La salida debería ser 1 si el número de entradas 1 en X y Y desde el restablecimiento es un múltiplo de 4, y 0 en otro caso.

A primera vista, usted puede pensar que la máquina necesita un número infinito de estados, puesto que cuenta entradas 1 a lo largo de un tiempo arbitrariamente largo. Sin embargo, ya que la salida indica el número de entradas recibidas *módulo 4*, cuatro estados son suficientes. Nombrémoslos S0-S3, donde S0 es el estado inicial y el número total de unos recibidos en S*i* es *i* módulo 4. La tabla 7-12 es la tabla de salida y de estado resultante.

Tabla 7-12
Tabla de salida y estado para la máquina de conteo de unos.

		XY				
Significado	***S***	***00***	***01***	***11***	***10***	***Z***
Obtención de cero unos (módulo 4)	S0	S0	S1	S2	S1	1
Obtención de un uno (módulo 4)	S1	S1	S2	S3	S2	0
Obtención de dos unos (módulo 4)	S2	S2	S3	S0	S3	0
Obtención de tres unos (módulo 4)	S3	S3	S0	S1	S0	0
		S*				

	XY				
Q1 Q2	00	01	11	10	Z
00	00	01	11	01	1
01	01	11	10	11	0
11	11	10	00	10	0
10	10	00	01	00	0
	Q1* Q2* o D1 D2				

Tabla 7-13
Tabla de salida y de transición/excitación para la máquina de conteo de unos.

La máquina de conteo de unos puede utilizar dos variables de estado para codificar sus cuatro estados, sin estados no utilizados. En este caso, existen solamente 4! posibles asignaciones de estados codificados para estados nombrados. Con todo, intentaremos solamente con una de ellas. Asignaremos estados codificados a los estados nombrados en orden de mapa de Karnaugh (00, 01, 11, 10) por dos razones: en esta tabla de estado, minimiza el número de variables de estado que cambian para la mayoría de las transiciones, simplificando potencialmente las ecuaciones de excitación; y simplifica la transferencia mecánica de información a los mapas de excitación.

Una tabla de transición/excitación basada en nuestra asignación de estado elegida se muestra en la figura 7-13. Como utilizamos flip-flops D, las tablas de transición y excitación son las mismas. Los mapas de Karnaugh correspondientes para D1 y D2 se muestran en la figura 5-57. Puesto que no hay estados sin utilizar, toda la información que necesitamos se encuentra en la tabla de excitación; no se requiere de elección entre los enfoques de riesgo mínimo y costo mínimo. Las ecuaciones de excitación pueden leerse de los mapas y la ecuación de salida puede leerse directamente de la tabla de transición/excitación:

$$D1 = Q2 \cdot X' \cdot Y + Q1' \cdot X \cdot Y + Q1 \cdot X' \cdot Y' + Q2 \cdot X \cdot Y'$$
$$D2 = Q1' \cdot X' \cdot Y + Q1' \cdot X \cdot Y' + Q2 \cdot X' \cdot Y' + Q2' \cdot X \cdot Y$$
$$Z = Q1' \cdot Q2'$$

A partir de estas ecuaciones se puede dibujar un diagrama lógico de flip-flops D y lógica de excitación AND-OR o NAND-NAND.

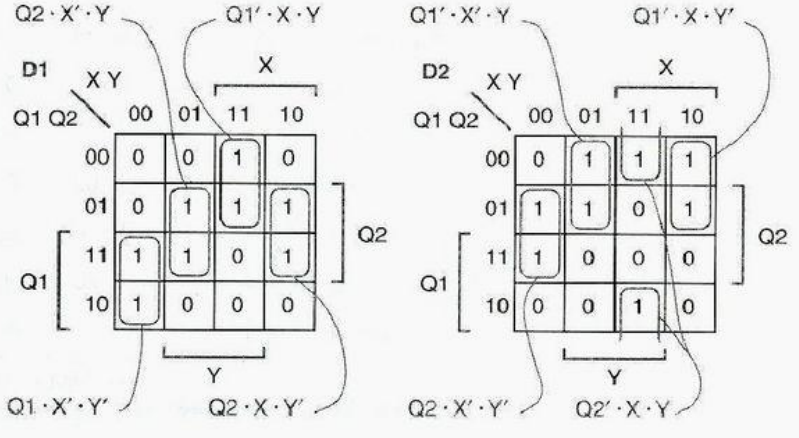


Figura 7-57
Mapas de excitación para entradas D1 y D2 en la máquina de conteo de unos.

El segundo ejemplo es una máquina de estado de "latch de combinación" que activa una salida de "apertura" cuando cierta secuencia de entrada binaria se recibe:

> Diseñe una máquina de estado sincrónica temporizada con una entrada, X, y dos salidas, UNLK y HINT. La salida UNLK debería ser 1 si sólo si X es 0 y la secuencia de entradas recibidas en X en los siete anteriores tics de reloj fue 0110111. La salida HINT debería ser 1 si y sólo si el valor actual de X es el correcto para mover la máquina más cerca de estar en el estado "abierto" (con UNLK = 1).

Debería ser evidente de la descripción con palabras que ésta es una máquina de Mealy. La salida UNLK depende tanto de la historia pasada de entradas como del valor actual de X, y HINT depende tanto del estado como del valor de X (en realidad, si la X actual produce HINT = 0, entonces el usuario informado querrá cambiar X antes del tic de reloj).

Una tabla de estado y salida para la latch de combinación se presenta en la tabla 7-14. En el estado inicial, A, suponemos que no hemos recibido entradas en la secuencia requerida: estamos buscando el primer 0 en la secuencia. Por lo tanto, mientras obtenemos entradas 1, permanecemos en el estado A, y nos movemos al estado B cuando recibimos un 0. En el estado B, buscamos por un 1. Si lo obtenemos, nos movemos a C; si no, podemos permanecer en B, puesto que el 0 que acabamos de recibir todavía puede resultar ser el primer 0 en la secuencia requerida. En cada estado sucesivo, nos movemos al estado siguiente si obtenemos la entrada correcta, y regresamos hacia A o B si obtenemos la entrada equivocada. Ocurre una excepción en el estado G; si obtenemos la entrada incorrecta (un 0) allí, las tres entradas previas todavía resultarán ser las primeras tres entradas de la secuencia requerida, de modo que regresamos al estado E en lugar de B. En el estado H, hemos recibido la secuencia requerida, de manera que establecemos UNLK a 1 si X es 0. En cada estado, establecemos HINT a 1 para el valor de X que nos mueve más cerca del estado H.

Tabla 7-14
Tabla de salida y estado para la máquina de latch de combinación.

		X	
Significado	***S***	***0***	***1***
Obtención de nada	A	B, 01	A, 00
Obtención de 0	B	B, 00	C, 01
Obtención de 01	C	B, 00	D, 01
Obtención de 011	D	E, 01	A, 00
Obtención de 0110	E	B, 00	F, 01
Obtención de 01101	F	B, 00	G, 01
Obtención de 011011	G	E, 00	H, 01
Obtención de 0110111	H	B, 11	A, 00
		S*, UNLK HINT	

	X	
Q1 Q2 Q3	0	1
000	001, 01	000, 00
001	001, 00	010, 01
010	001, 00	011, 01
011	100, 01	000, 00
100	001, 00	101, 01
101	001, 00	110, 01
110	100, 00	111, 01
111	001, 11	000, 00
	Q1* Q2* Q3*, UNLK HINT	

Tabla 7-15
Tabla de transición/ excitación para la máquina de cerradura de combinación.

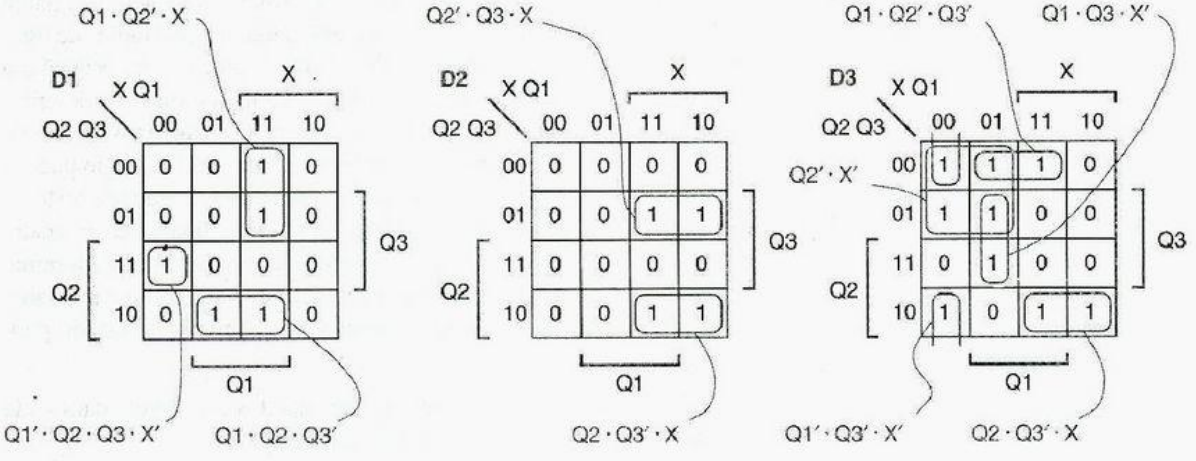
La combinación de los ocho estados de la cerradura de combinación puede ser codificada con tres variables de estado, sin dejar estados sin utilizar. Existen 8! asignaciones de estado para elegir. Para mantener las cosas sencillas, utilizaremos la más simple, y asignaremos los estados en orden de conteo binario, produciendo la tabla de transición/excitación 7-15. Los mapas de Karnaugh correspondientes para D1, D2 y D3 se muestran en la figura 7-58. Las ecuaciones de excitación se pueden leer a partir de los mapas:

$$D1 = Q1 \cdot Q2' \cdot X + Q1' \cdot Q2 \cdot Q3 \cdot X' + Q1 \cdot Q2 \cdot Q3'$$

$$D2 = Q2' \cdot Q3 \cdot X + Q2 \cdot Q3' \cdot X$$

$$D3 = Q1 \cdot Q2' \cdot Q3' + Q1 \cdot Q3 \cdot X' + Q2' \cdot X' + Q3' \cdot Q1' \cdot X' + Q2 \cdot Q3' \cdot X$$

Figura 7-58 Mapas de excitación para D1, D2 y D3 en la máquina de cerradura de combinación.



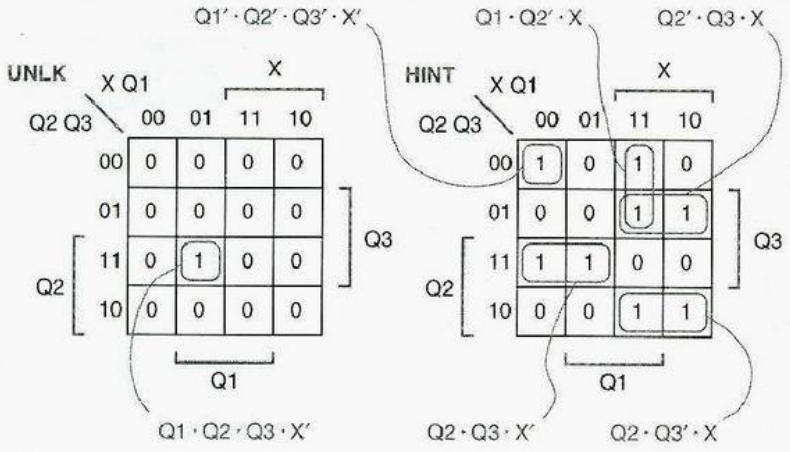


Figura 7-59
Mapas de Karnaugh para funciones de salida UNLK y HINT en la máquina de cerradura de combinación.

Los valores de salida son transferidos de las tablas de transición/excitación y de salida a otro conjunto de mapas en la figura 7-59. Las ecuaciones de salida correspondientes son:

$$UNLK = Q1 \cdot Q2 \cdot Q3 \cdot X'$$

$$HINT = Q1' \cdot Q2' \cdot Q3' \cdot X' + Q1 \cdot Q2' \cdot X + Q2' \cdot Q3 \cdot X + Q2 \cdot Q3 \cdot X' + Q2 \cdot Q3' \cdot X$$

Nótese que algunos términos de producto se repiten en las ecuaciones de excitación y de salida, produciendo pequeños ahorros en el costo de la realización AND-OR. Si fuéramos a través del problema de efectuar una minimización de salida múltiple formal de todas las cinco funciones de excitación y salida, podríamos ahorrar dos compuertas más (véase el ejercicio 7.55).

7.5 Diseño de máquinas de estado que utilizan diagramas de estado

Aparte de planear la arquitectura total de un sistema digital, el diseño de máquinas de estado es probablemente la tarea más creativa de un diseñador digital. La mayoría de las personas gustan de tomar un enfoque gráfico para el diseño: usted probablemente habrá resuelto muchos problemas solamente garabateando. Por esa razón, los diagramas de estado se emplean con frecuencia para diseñar máquinas de estado de dimensiones pequeñas a medianas. En esta sección daremos ejemplos de diseño de diagramas de estado y describiremos un procedimiento simple para sintetizar circuitos a partir de los diagramas de estado. Este procedimiento es la base del método utilizado por herramientas de CAD que pueden sintetizar la lógica a partir de "diagramas de estado" gráficos o incluso basados en texto.

El diseño de un diagrama de estado es muy semejante a diseñar una tabla de estado, lo que, como demostramos en la sección 7.4.1, es muy semejante a escribir un programa. Sin embargo, existe una diferencia fundamental entre un diagrama de estado y una tabla de estado, una diferencia que hace más sencillo el diseño del diagrama de estado pero también más propenso a errores:

- Una tabla de estado es una lista exhaustiva de los estados siguientes para cada combinación estado/entrada. No es posible la ambigüedad.

- Un diagrama de estado contiene un conjunto de arcos etiquetados con expresiones de transición. Aun cuando existen muchas entradas, solamente una expresión de transición es requerida por arco. Sin embargo, cuando un diagrama de estado se construye, no hay garantía de que las expresiones de transición escritas en los arcos que dejan un estado particular cubran la totalidad de las combinaciones de entrada exactamente una vez.

diagrama de estado ambiguo

En un diagrama de estado cuya construcción es inadecuada (*ambigua*), el estado siguiente de algunas combinaciones de entrada puede no estar especificado, lo que es indeseable, pero se pueden especificar varios estados siguientes para otras combinaciones de entrada, lo cual es un error. Se debe tener mucha precaución en el diseño de los diagramas de estado; ofreceremos varios ejemplos.

Nuestro primer ejemplo es una máquina de estado que controla las luces traseras o "calaveras" de un Ford Thunderbird modelo 1965, ilustrado en la figura 7-60. Se tienen tres luces de cada lado, y para las vueltas funcionan en secuencia para mostrar la dirección en que se da la vuelta, como se ilustra en la figura 7-61. La máquina de estado tiene dos señales de entrada, LEFT y RIGHT ("izquierda" y "derecha", respectivamente), que llevan la petición del conductor para dar una vuelta a la izquierda o a la derecha. También tienen una entrada de luces intermitentes de emergencia, HAZ, que solicita que las luces traseras se hagan funcionar en el modo de peligro: todas las seis luces parpadean al unísono. También supondremos la existencia de una señal de reloj de funcionamiento libre cuya frecuencia es igual a la velocidad de destello intermitente deseado para las luces.

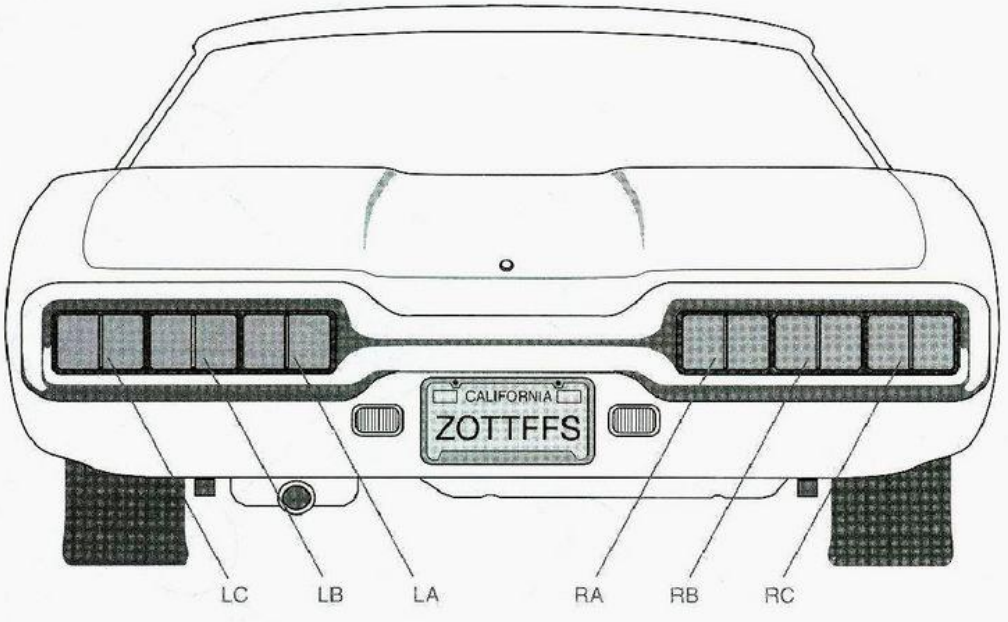


Figura 7-60
Luces traseras del Thunderbird.

¿DE QUIÉN ES LA PARTE TRASERA? En realidad, la figura 7-60 tiene un aspecto más parecido al extremo posterior de un Mercury Capri, que también tiene luces traseras secuenciales.

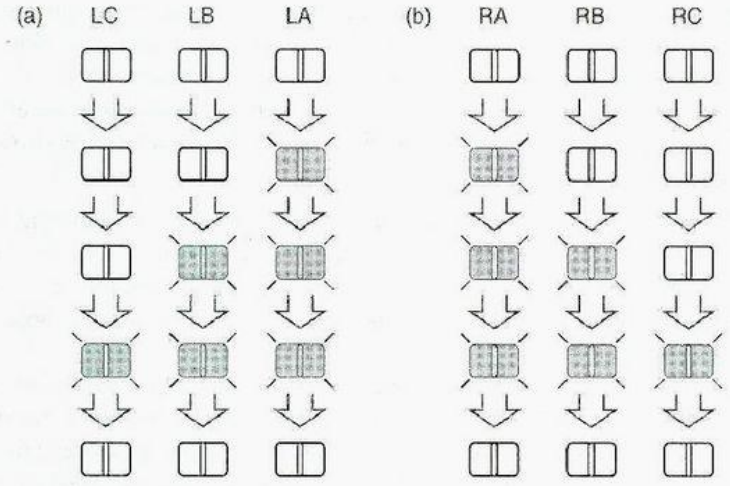


Figura 7-61
Secuencia de destello para las luces traseras del Thunderbird: (a) giro a la izquierda; (b) giro a la derecha.

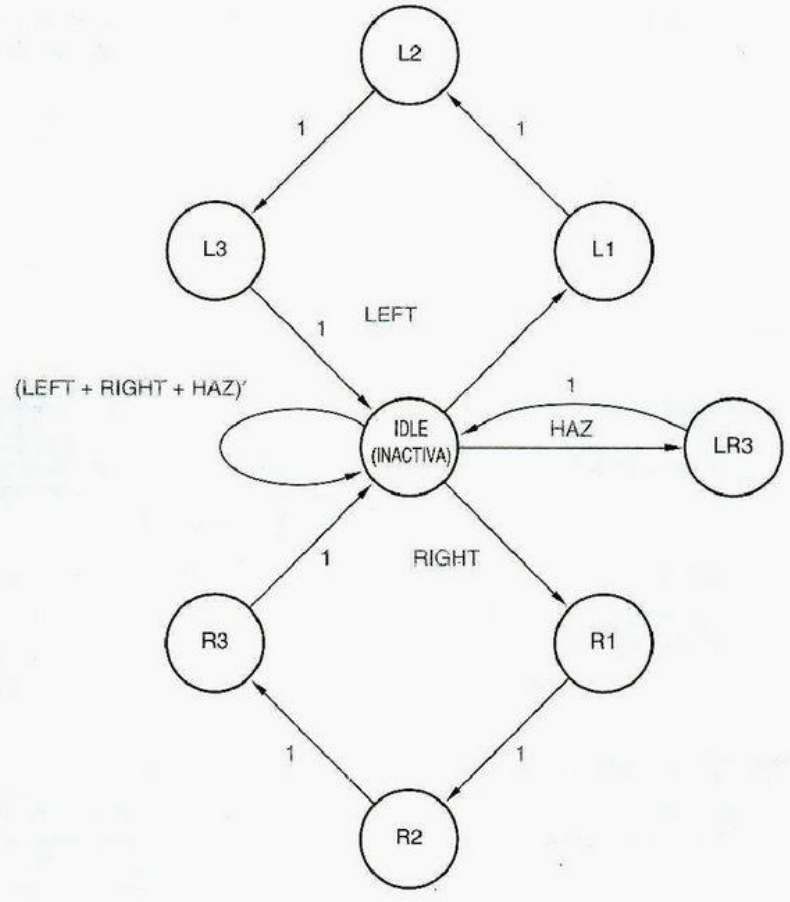


Figura 7-62
Diagrama de estado inicial y tabla de salida para las luces traseras del Thunderbird.

Tabla de salida

Estado	LC	LB	LA	RA	RB	RC
IDLE	0	0	0	0	0	0
L1	0	0	1	0	0	0
L2	0	1	1	0	0	0
L3	1	1	1	0	0	0
R1	0	0	0	1	0	0
R2	0	0	0	1	1	0
R3	0	0	0	1	1	1
LR3	1	1	1	1	1	1

Dados los requerimientos anteriores, podemos diseñar una máquina de estado sincrónica temporizada para controlar las luces traseras del automóvil. Diseñaremos una máquina de Moore de modo que el estado sólo determine cuáles luces se encuentran encendidas y cuáles se encuentran apagadas. Para una vuelta a la izquierda, la máquina debería recorrer el ciclo a través de cuatro estados en los cuales las luces de la derecha se encuentran apagadas y 0, 1, 2 o 3 de las luces izquierdas se encuentran encendidas. Del mismo modo para una vuelta a la derecha, debería recorrer el ciclo a través de cuatro estados en los cuales las luces de la izquierda encuentran apagadas y 0, 1, 2 o 3 de las luces derechas se encuentran encendidas. En el modo intermitente, solamente y se requieren dos estados: todas las luces encendidas o todas las luces apagadas.

La figura 7-62 muestra nuestro primer corte para un diagrama de estado para la máquina. Se define un estado común IDLE para el cual todas las luces se encuentran apagadas. Cuando se solicita una vuelta a la izquierda, la máquina pasa a través de tres estados en los cuales 1, 2 y 3 de las luces izquierdas están encendidas, y luego va de regreso a IDLE; las vueltas a la derecha funcionan de manera similar. En el modo intermitente, la máquina entra en un ciclo que va del estado IDLE al estado en el que todas las seis luces se encuentran encendidas. Puesto que hay tantas salidas, hemos incluido una tabla de salida separada, más que escribir valores de salida en el diagrama de estado. Aun sin asignar estados codificados a los estados nombrados, podemos escribir las ecuaciones de salida a partir de la tabla de salida, si permitimos que cada nombre de estado represente una expresión lógica que sea 1 solamente en ese estado:

$$LA = L1 + L2 + L3 + LR3 \qquad RA = R1 + R2 + R3 + LR3$$
$$LB = L2 + L3 + LR3 \qquad RB = R2 + R3 + LR3$$
$$LC = L3 + LR3 \qquad RC = R3 + LR3$$

Existe un gran problema con el diagrama de estado de la figura 7-62: no maneja apropiadamente entradas múltiples afirmadas simultáneamente. Por ejemplo, ¿qué ocurre en el estado IDLE si tanto LEFT como HAZ son afirmadas? De acuerdo al diagrama de estado, la máquina se va a dos estados, L1 y LR3, lo cual es imposible. En realidad, la máquina tendría solamente un estado siguiente, que sería L1, LR3 o un tercer estado totalmente sin relación (y posiblemente sin utilizar), dependiendo de los detalles de la realización de la máquina de estado (por ejemplo, véase el ejercicio 7.58).

El problema se arregla en la figura 7-63, donde hemos dado prioridad a la entrada HAZ. También, tratamos LEFT y RIGHT afirmadas en forma simultánea como una solicitud de luces intermitentes, puesto que el conductor evidentemente se encuentra confundido y necesita ayuda.

El nuevo diagrama de estado no es ambiguo porque las expresiones de transición en los arcos que dejan cada estado son mutuamente exclusivas y totalmente inclusivas. Es decir, para cada estado, ningún par de expresiones son 1 para la misma combinación de entrada, y alguna expresión es 1 para cada combinación de entrada. Esto puede confirmarse algebraicamente para éste y cualquier otro diagrama de estado mediante la realización de dos pasos:

1. *Exclusión mutua.* Para cada estado, muestre que el producto lógico de cada par posible de expresiones de transición en los arcos que dejan ese estado es 0. Si existen n arcos, entonces existen $n(n-1)/2$ productos lógicos por evaluar. *exclusión mutua*
2. *Inclusión total.* Para cada estado, muestre que la suma lógica de las expresiones de transición en todos los arcos que dejan ese estado es 1. *inclusión total*

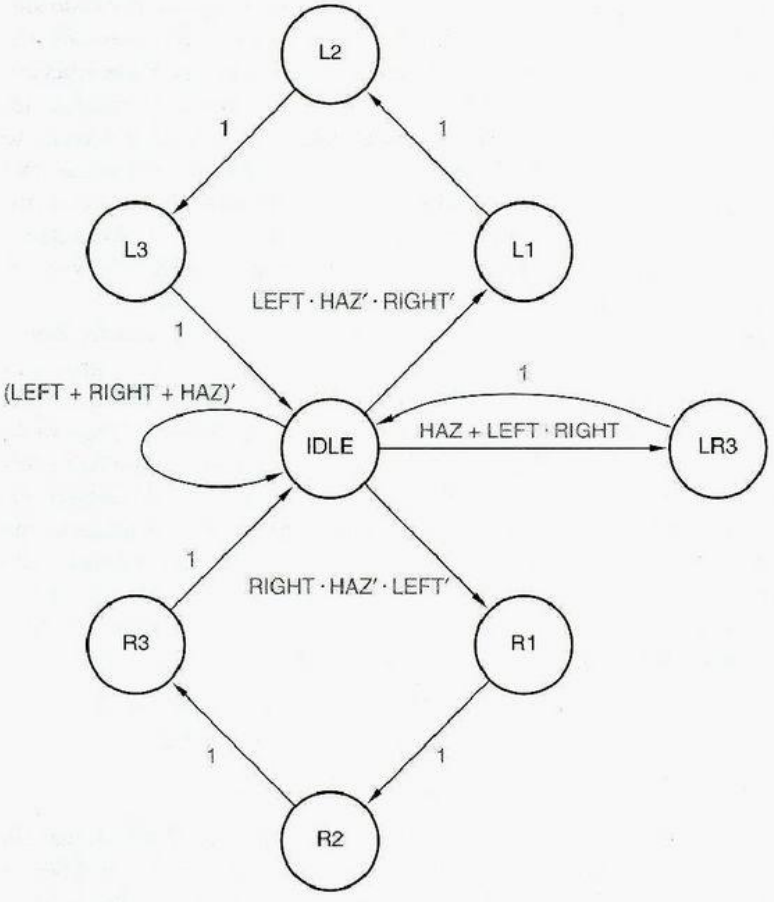


Figura 7-63
Diagrama de estado corregido para las luces traseras del Thunderbird.

Si hay muchas transiciones dejando cada estado, estos pasos, especialmente el primero, son muy difíciles de efectuar. Sin embargo, las máquinas de estado típicas, incluso aquellas con montones de estados y entradas, no tienen muchas transiciones dejando cada estado, puesto que la mayoría de los diseñadores no pueden idear tales máquinas complejas en primer lugar. Aquí es donde se presenta el compromiso entre diseño de tabla de estado y diagrama de estado. En el diseño de la tabla de estado, los pasos anteriores no son requeridos, porque la estructura de una tabla de estado garantiza la exclusión mutua y la inclusión total. Pero si hay una gran cantidad de entradas, la tabla de estado tendrá *bastantes* columnas.

Verificar que un diagrama de estado no sea ambiguo puede ser difícil en principio, pero no es demasiado malo en la práctica para diagramas de estado pequeños. En la figura 7-63, la mayoría de los estados tienen un solo arco con una expresión de transición de 1, de modo que la verificación es trivial. El trabajo real se necesita solamente para verificar el estado IDLE, del que parten cuatro transiciones. Esto puede hacerse en una hoja de papel para borradores numerando las ocho combinaciones de las tres entradas y señalando las combinaciones cubiertas por cada expresión de transición. Cada combinación debería tener exactamente una marca. Como otro ejemplo, considere los diagramas de estado en las figura 7-44 y 7-46; ambas pueden verificarse mentalmente.

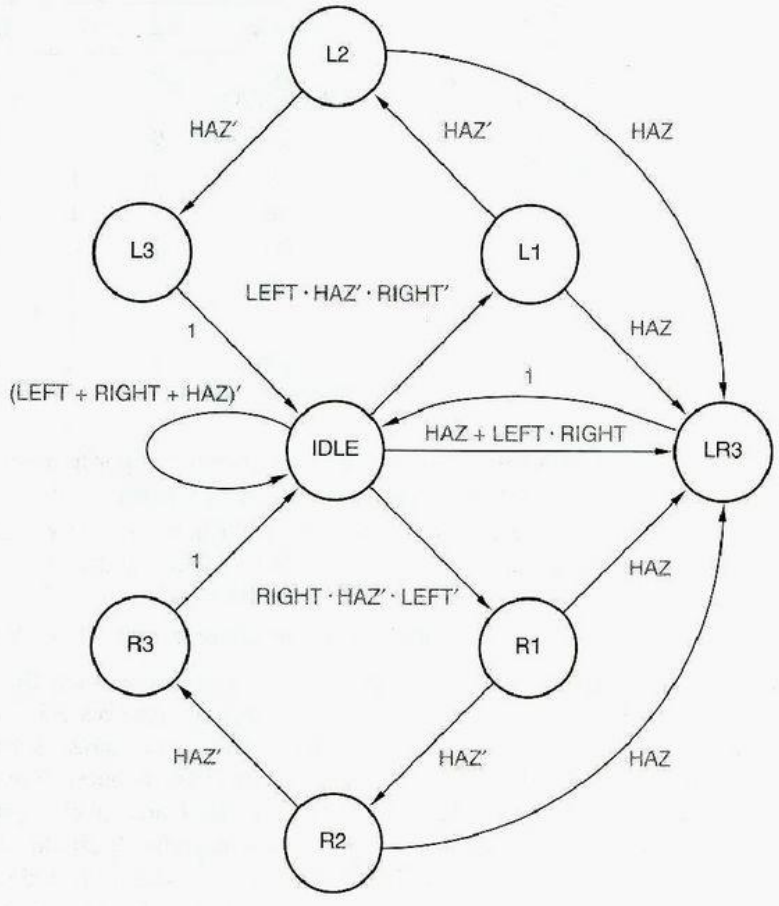


Figura 7-64
Diagrama de estado mejorado para las luces traseras del Thunderbird.

Regresando a la máquina de las luces traseras del Thunderbird, podemos sintetizar un circuito a partir del diagrama de estado si lo deseamos. Sin embargo, si queremos cambiar el comportamiento de la máquina ahora es tiempo de hacerlo, antes de que hagamos todo el trabajo de sintetizar un circuito. En particular, note que una vez que ha comenzado un ciclo de vuelta a la izquierda o a la derecha, el diagrama de estado en la figura 7-63 permite que el ciclo se ejecute hasta completarse, incluso si HAZ es afirmada. Mientras que esto puede tener cierto atractivo estético, sería más seguro para los ocupantes del automóvil tener la máquina entrando en modo de luces intermitentes tan pronto como fuera posible. El diagrama de estado está modificado para proporcionar este comportamiento en la figura 7-64.

Ahora estamos listos para sintetizar un circuito para la máquina del Thunderbird. El diagrama de estado tiene ocho estados, de modo que necesitaremos un mínimo de tres flip-flops para codificar los estados. Obviamente, son posibles muchas asignaciones de estado (8! para ser exactos); utilizaremos la de la tabla 7-16 por las siguientes razones:

1. Un estado inicial (inactivo) de 000 es compatible con la mayoría de los flip-flops y registros, que son fácilmente inicializados al estado 0.
2. Dos variables de estado, Q1 y Q0, se utilizan para "conteo" en secuencia de código Gray para el ciclo de la vuelta izquierda (IDLE→L1→L2→L3→IDLE). Esto

Tabla 7-16
Asignación de estado para la máquina de estado de las luces traseras del Thunderbird.

Estado	*Q2*	*Q1*	*Q0*
IDLE (INACTIVO)	0	0	0
L1	0	0	1
L2	0	1	1
L3	0	1	0
R1	1	0	1
R2	1	1	1
R3	1	1	0
LR3	1	0	0

minimiza el número de cambios de estado-variables por transición de estado, lo que a menudo puede simplificar la lógica de excitación.

3. Debido a la simetría en el diagrama de estado, la misma secuencia en Q1 y Q0 es empleada para "conteo" durante un ciclo de vuelta a la derecha, mientras que Q2 se utiliza para distinguir entre izquierda y derecha.
4. La combinación estado-variable restante se emplea para el estado LR3.

El siguiente paso es escribir una especie de tabla de transición. Sin embargo, debemos utilizar un formato diferente de las tablas de transición de la sección 7.4.4, porque las transiciones en un diagrama de estado son especificadas por expresiones más que por una tabulación exhaustiva de estados siguientes. Llamaremos al nuevo formato una *lista de transición*, porque tiene un renglón por cada transición o arco en el diagrama de estado.

lista de transición

La tabla 7-17 es la lista de transición para el diagrama de estado de la figura 7-64 y la asignación de estado de la tabla 7-16. Cada renglón contiene el estado actual, el estado siguiente y la expresión de transición para un arco en el diagrama de estado. Ambas versiones, nombrada y codificada, del estado actual y el estado siguiente se muestran. Los estados nombrados son útiles para propósitos de referencia, mientras que los estados codificados se utilizan para desarrollar ecuaciones de transición.

Una vez que tenemos una lista de transición, el resto de los pasos de síntesis son más o menos cuestión de "arrastrar el lápiz". Los procedimientos de síntesis se describen en la sección 7.6. Aunque estos procedimientos pueden aplicarse en forma manual, por lo regular se integran en un paquete de software de CAD; de este modo, la sección 7.6 puede ayudarnos a comprender qué es lo que ocurre (o qué es lo que va mal) en su paquete favorito de CAD.

También encontramos un paso de "trabajo duro" en esta sección: hallar las ambigüedades en los diagramas de estado. Aun cuando el procedimiento que discutimos puede ser fácilmente automatizado, pocos, si es que algunos, programas de CAD realizan este paso de esta manera. Por ejemplo, una herramienta de captura de "diagrama de estado" silenciosamente elimina transiciones duplicadas y se va al estado codificado "00...00" para transiciones perdidas, sin prevenir al usuario. De este modo, en la mayoría de los ambientes de diseño, el diseñador es responsable de escribir una descripción de máquina de estado que no sea ambigua. Los lenguajes de descripción de máquina de estado al final de este capítulo proporcionan una buena manera de hacer esto.

S	Q2	Q1	Q0	Expresión de transición	S*	Q2*	Q1*	Q0*
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

Tabla 7-17 Lista de transición para la máquina de estado de las luces traseras del Thunderbird.

*7.6 Síntesis de una máquina de estado que utiliza listas de transición

Una vez que se ha diseñado el diagrama de una máquina de estado y se ha efectuado una asignación de estado, la parte creativa del proceso de diseño se encuentra más o menos acabada. El resto del procedimiento de síntesis puede ser llevado a cabo por medio de programas de CAD.

Como mostramos en la sección anterior, puede construirse una lista de transición a partir del diagrama de estado y la asignación de estado de una máquina. Esta sección muestra cómo sintetizar una máquina de estado a partir de su lista de transición. También profundiza en algunas de las opciones y matices del diseño de la máquina de estado que utiliza listas de transición. Aunque este material es útil para sintetizar máquinas a mano, su propósito principal es ayudar a entender la operación interna y las peculiaridades externas de los lenguajes y programas de CAD que tratan con máquinas de estado.

*7.6.1 Ecuaciones de transición

El primer paso para sintetizar una máquina de estado a partir de una lista de transición es desarrollar un conjunto de ecuaciones de transición que definan cada variable V* de estado siguiente en términos del estado y entrada actuales. La lista de transición puede ser visualizada como una especie de tabla de verdad híbrida en la cual las combinaciones

* Esta sección y todas sus subsecciones son opcionales.

variable de estado para el estado actual se numeran de manera explícita y las combinaciones de entrada se numeran algebraicamente. Si leemos hacia abajo una columna de V* en una lista de transición, encontramos una secuencia de ceros y unos, indicando el valor de V* para varias (todas si hemos hecho todo correctamente) combinaciones estado/entrada.

Una ecuación de transición para una variable V* de estado siguiente puede escribirse utilizando una especie de suma canónica híbrida:

$$V* = \sum_{\text{renglones de lista de transición donde } V* = 1} (\text{p término de transición})$$

p término de transición

Es decir, la ecuación de transición tiene un "p término de transición" para cada renglón de la lista de transición que contiene un 1 en la columna V*. Un *p término de transición* del renglón es el producto del minitérmino de estado actual y la expresión de transición.

Basados en la lista de transición de la tabla 7-17, la ecuación de transición para Q2*, en la máquina del Thunderbird, puede escribirse como la suma de ocho p términos:

$$\begin{aligned} Q2* = {} & Q2' \cdot Q1' \cdot Q0' \cdot (HAZ + LEFT \cdot RIGHT) \\ & + Q2' \cdot Q1' \cdot Q0' \cdot (RIGHT \cdot HAZ' \cdot LEFT') \\ & + Q2' \cdot Q1' \cdot Q0 \cdot (HAZ) \\ & + Q2' \cdot Q1 \cdot Q0 \cdot (HAZ) \\ & + Q2 \cdot Q1' \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1' \cdot Q0 \cdot (HAZ) \\ & + Q2 \cdot Q1 \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1 \cdot Q0 \cdot (HAZ) \end{aligned}$$

Algunas manipulaciones algebraicas directas conducen a una ecuación de transición simplificada que combina los primeros dos, los segundos dos y los cuatro últimos p términos anteriores:

$$\begin{aligned} Q2* = {} & Q2' \cdot Q1' \cdot Q0' \cdot (HAZ + RIGHT) \\ & + Q2' \cdot Q0 \cdot (HAZ) \\ & + Q2 \cdot Q0 \end{aligned}$$

Las ecuaciones de transición para Q1* y Q0* pueden obtenerse de una manera similar:

$$\begin{aligned} Q1* = {} & Q2' \cdot Q1' \cdot Q0 \cdot (HAZ') \\ & + Q2' \cdot Q1 \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1' \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1 \cdot Q0 \cdot (HAZ') \\ = {} & Q0 \cdot HAZ' \\ Q0* = {} & Q2' \cdot Q1' \cdot Q0' \cdot (LEFT \cdot HAZ' \cdot RIGHT') \\ & + Q2' \cdot Q1' \cdot Q0' \cdot (RIGHT \cdot HAZ' \cdot LEFT') \\ & + Q2' \cdot Q1' \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1' \cdot Q0 \cdot (HAZ') \\ = {} & Q2' \cdot Q1' \cdot Q0' \cdot HAZ' \cdot (LEFT \oplus RIGHT) + Q1' \cdot Q0 \cdot HAZ' \end{aligned}$$

Excepto para Q1*, no hay garantía de que las ecuaciones de transición anteriores sean en algún sentido mínimas; de hecho, las expresiones para Q2* y Q0* no están

incluso en forma estándar de producto de sumas. Las ecuaciones simplificadas, o las originales sin simplificar, proporcionan meramente un punto de partida preciso para cualquier método de diseño combinacional que usted puede elegir para sintetizar la lógica de excitación para la máquina de estado: ad hoc, NAND-NAND, basada en MSI o cualquier otra. En un diseño basado en PLD, simplemente se podrían agregar las ecuaciones a un programa ABEL y dejar que el compilador calcule las expresiones de suma de producto mínimas para el arreglo AND-OR del PLD.

*7.6.2 Ecuaciones de excitación

Mientras estamos en el tema de la lógica de excitación, note que hasta ahora hemos derivado solamente *ecuaciones de transición*, no *ecuaciones de excitación*. Sin embargo, si utilizamos flip-flops D como los elementos de memoria en nuestras máquinas de estado, entonces las ecuaciones de excitación son fáciles de obtener a partir de las ecuaciones de transición, puesto que la ecuación característica de un flip-flop D es Q* = D. Por lo tanto, si la ecuación de transición para una variable de estado Qi* es

$$Qi* = \text{expresión}$$

entonces la ecuación de excitación para la correspondiente entrada de flip-flop D es

$$Di = \text{expresión}$$

Las ecuaciones de excitación eficientes para otros tipos de flip-flops, especialmente los del tipo J-K, no son fáciles de obtener (vea el ejercicio 7.63). Por esa razón, la gran mayoría de diseños discretos, basados en PLD y basados en ASIC de máquina de estado emplean flip-flops D.

*7.6.3 Variaciones sobre el esquema

Existen otras maneras de obtener ecuaciones de transición y excitación a partir de una lista de transición. Si la columna para una variable particular de estado siguiente contiene menos ceros que unos, puede ser ventajoso escribir esa ecuación de transición de la variable en términos de los ceros en su columna. Es decir, escribiríamos

$$V*' = \sum_{\text{renglones de lista de transición donde } V* = 0} (\text{p término de transición})$$

Esto es, V*' es 1 para todos los p términos para los cuales V* es 0. De este modo, una ecuación de transición para Q2*' puede escribirse como la suma de siete p términos:

$$\begin{aligned} Q2*' &= Q2' \cdot Q1' \cdot Q0' \cdot ((LEFT + RIGHT + HAZ)') \\ &\quad + Q2' \cdot Q1' \cdot Q0' \cdot (LEFT \cdot HAZ' \cdot RIGHT') \\ &\quad + Q2' \cdot Q1' \cdot Q0 \cdot (HAZ') \\ &\quad + Q2' \cdot Q1 \cdot Q0 \cdot (HAZ') \\ &\quad + Q2' \cdot Q1 \cdot Q0' \cdot (1) \\ &\quad + Q2 \cdot Q1 \cdot Q0' \cdot (1) \\ &\quad + Q2 \cdot Q1' \cdot Q0' \cdot (1) \\ &= Q2' \cdot Q1' \cdot Q0' \cdot HAZ' \cdot RIGHT' + Q2' \cdot Q0 \cdot HAZ' + Q1 \cdot Q0' + Q2 \cdot Q0' \end{aligned}$$

Para obtener una ecuación para Q2*, simplemente complementamos ambos lados de la ecuación reducida.

Para obtener una expresión para una variable V* de estado siguiente directamente, empleando los ceros en la lista de transición, podemos complementar el lado derecho de la ecuación general de V*' aplicando el teorema de DeMorgan, obteniendo una especie de producto canónico híbrido:

$$V^* = \prod_{\text{renglones de lista de transición donde } V^* = 0} (\text{s término de transición})$$

s término de transición

Aquí, un *s término de transición* del renglón es la suma del maxitérmino para el estado actual y el complemento de la expresión de transición. Si la expresión de transición es un término de producto simple, entonces su complemento es una suma, y la ecuación de transición expresa V* en forma de producto de sumas.

*7.6.4 Realización de la máquina de estado

Una vez que se tienen las ecuaciones de excitación para una máquina de estado, todo lo que resta es un problema de diseño lógico combinacional de salida múltiple. Naturalmente, existen muchas maneras de realizar lógica combinacional de las ecuaciones, pero la manera más fácil es únicamente introducirlas en un programa ABEL o VHDL y utilizar el compilador para sintetizar una realización PLD, FPGA o ASIC.

Los PLD combinacionales, tales como el PAL16L8 y el GAL16V8 que se estudiaron en la sección 5.3, pueden emplearse para realizar ecuaciones de excitación hasta un cierto número de entradas, salidas y términos de producto. Aún mejor, en la sección 8.3 presentaremos PLD secuenciales que incluyen flip-flops D en el mismo chip con el arreglo combinacional AND-OR. Para un número dado de terminales de entrada y salida de PLD, estos PLD secuenciales pueden realizar máquinas de estado más grandes que sus homólogos combinacionales, debido a que las señales de excitación nunca tienen que salir del chip. En la sección 9.1.3, mostraremos cómo realizar la máquina de las luces traseras del Thunderbird en un PLD secuencial.

*7.7 Otro ejemplo de diseño de máquina de estado

Esta sección ofrece un ejemplo más de diseño de máquina de estado utilizando un diagrama de estado. El ejemplo proporciona un fundamento para discusión adicional de unos cuantos temas: estados sin utilizar, asignaciones de estado codificados por salida y codificaciones de estado "sin importancia".

7.7.1 El juego de las adivinanzas

Nuestro ejemplo final de máquina de estado es un "juego de adivinanzas" que puede construirse como un entretenido proyecto de laboratorio:

Diseñe una máquina de estado sincrónica temporizada con cuatro entradas, G1-G4, que se encuentren conectadas a unos botones. La máquina tiene cuatro salidas, L1-L4, conectadas a lámparas o LED localizados cerca de los botones numerados. También hay una salida ERR conectada a una lámpara roja. En funcionamiento normal las salidas L1-L4 exhiben un patrón 1 de 4. En cada tic de reloj, el patrón es alternado en una posición; la frecuencia de reloj es de aproximadamente 4 Hz.

Las adivinanzas se efectúan presionando un botón, lo que afirma una entrada Gi. Cuando una entrada Gi es afirmada, la salida ERR se afirma si el botón "equivocado" fue presionado, es decir, si la entrada Gi detectada en el tic de reloj no tiene el mismo número de la salida de la lámpara que fue afirmada antes del tic de reloj. Una vez que se hace una adivinanza, el juego se detiene y la salida ERR mantiene el mismo valor para uno o más tics de reloj hasta que la entrada Gi es negada, y luego el juego se reanuda.

Evidentemente, tendremos que proporcionar cuatro estados, uno para cada posición del patrón de alternancia, y necesitaremos por lo menos un estado para indicar que el juego se ha detenido. Un diagrama de estado posible se muestra en la figura 7-65. La máquina pasa cíclicamente a través de los estados S1-S4 mientras que ninguna entrada Gi sea afirmada, y se va al estado STOP cuando se efectúa una adivinanza. Cada salida Li es afirmada en el estado numerado correspondiente.

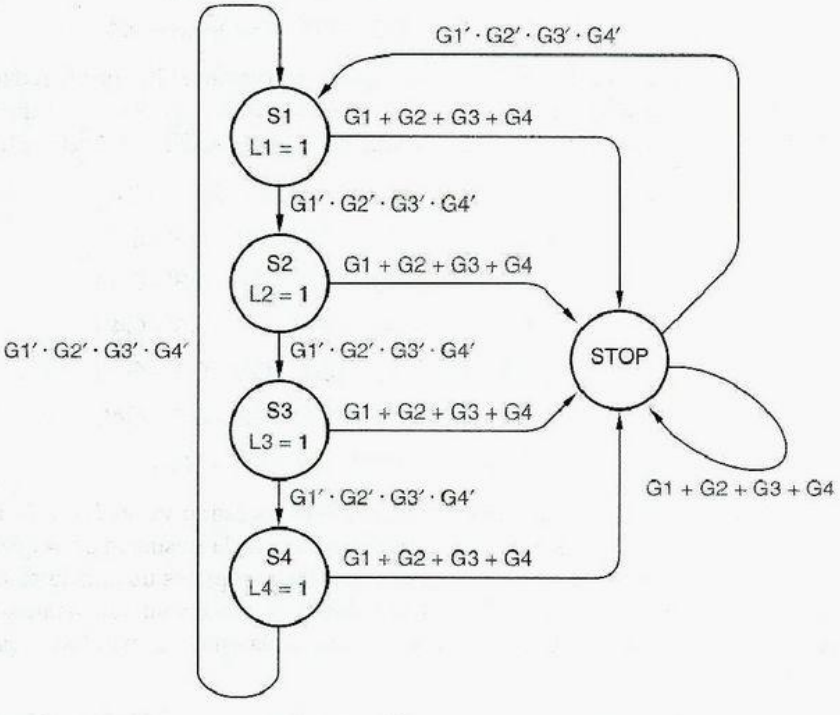


Figura 7-65
Primer intento para un diagrama de estado para el juego de adivinanzas.

El único problema con este diagrama de estado es que no "recuerda" en el estado STOP si la adivinanza fue correcta, de modo que no hay manera de controlar la salida ERR. Este problema se arregla en la figura 7-66, la cual tiene dos estados "de paro", SOK y SERR. Con una adivinanza incorrecta, la máquina se va a SERR, donde ERR se afirma; de otra manera, se va a SOK. Aunque la descripción en palabras de la máquina no lo requiere, el diagrama de estado está diseñado para ir a SERR incluso si el usuario intenta engañarla presionando dos o más botones simultáneamente, o cambiando las adivinanzas mientras se detiene.

Una lista de transición correspondiente al diagrama de estado se muestra en la tabla 7-18, que utiliza un estado binario simple de 3 bits codificado con orden de código Gray para el ciclo S1-S4. Las ecuaciones de transición para Q1* y Q0* pueden obtenerse de la tabla como se ve a continuación:

$$
\begin{aligned}
Q1* &= Q2' \cdot Q1' \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2' \cdot Q1 \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&= Q2' \cdot Q0 \cdot G1' \cdot G2' \cdot G3' \cdot G4' \\
Q0* &= Q2' \cdot Q1' \cdot Q0' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2' \cdot Q1' \cdot Q0' \cdot (G2 + G3 + G4) \\
&\quad + Q2' \cdot Q1' \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2' \cdot Q1' \cdot Q0 \cdot (G1 + G3 + G4) \\
&\quad + Q2' \cdot Q1 \cdot Q0 \cdot (G1 + G2 + G4) \\
&\quad + Q2' \cdot Q1 \cdot Q0' \cdot (G1 + G2 + G3) \\
&\quad + Q2 \cdot Q1' \cdot Q0 \cdot (G1 + G2 + G3 + G4)
\end{aligned}
$$

Al aplicar un programa de minimización lógica, la expresión Q0* puede reducirse a 11 términos de producto en forma de suma de productos de dos niveles. Una expresión para Q2* se formula mejor en términos de los ceros en la columna Q2* de la lista de transición:

$$
\begin{aligned}
Q2*' &= Q2' \cdot Q1' \cdot Q0' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2' \cdot Q1' \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2' \cdot Q1 \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2' \cdot Q1 \cdot Q0' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2 \cdot Q1' \cdot Q0' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + Q2 \cdot Q1' \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&= (Q2' + Q1') \cdot (G1' \cdot G2' \cdot G3' \cdot G4')
\end{aligned}
$$

Las últimas cinco columnas de la tabla 7-18 muestran valores de salida. De esta forma, las ecuaciones de salida pueden desarrollarse de la misma manera que las ecuaciones de transición. Sin embargo, puesto que este ejemplo es una máquina de Moore, las salidas son independientes de las expresiones de transición: solamente un renglón de la lista de transición debe ser considerado para cada estado actual. Las ecuaciones de salida son

$$
\begin{aligned}
L1 &= Q2' \cdot Q1' \cdot Q0' \qquad & L3 &= Q2' \cdot Q1 \cdot Q0 \qquad & ERR &= Q2 \cdot Q1' \cdot Q0 \\
L2 &= Q2' \cdot Q1' \cdot Q0 \qquad & L4 &= Q2' \cdot Q1 \cdot Q0'
\end{aligned}
$$

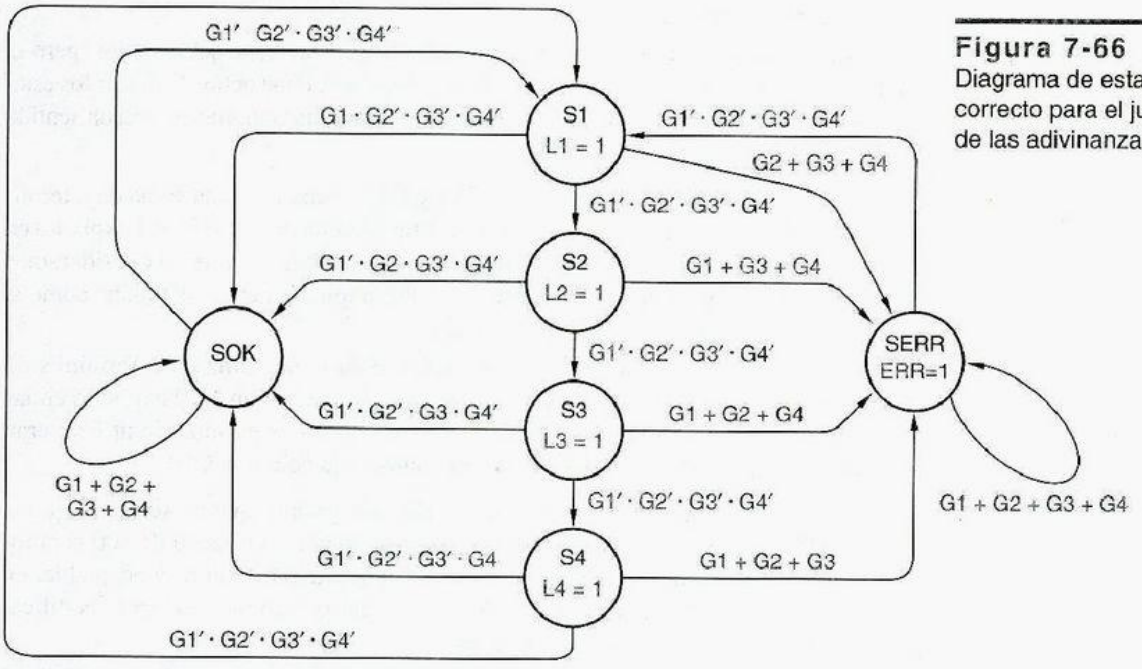


Figura 7-66
Diagrama de estado correcto para el juego de las adivinanzas.

Tabla 7-18 Lista de transición para la máquina del juego de las adivinanzas.

Estado actual					*Estado siguiente*				*Salida*				
S	***Q2***	***Q1***	***Q0***	***Expresión de transición***	***S****	***Q2****	***Q1****	***Q0****	***L1***	***L2***	***L3***	***L4***	***ERR***
S1	0	0	0	G1′ · G2′ · G3′ · G4′	S2	0	0	1	1	0	0	0	0
S1	0	0	0	G1 · G2′ · G3′ · G4′	SOK	1	0	0	1	0	0	0	0
S1	0	0	0	G2 + G3 + G4	SERR	1	0	1	1	0	0	0	0
S2	0	0	1	G1′ · G2′ · G3′ · G4′	S3	0	1	1	0	1	0	0	0
S2	0	0	1	G1′ · G2 · G3′ · G4′	SOK	1	0	0	0	1	0	0	0
S2	0	0	1	G1 + G3 + G4	SERR	1	0	1	0	1	0	0	0
S3	0	1	1	G1′ · G2′ · G3′ · G4′	S4	0	1	0	0	0	1	0	0
S3	0	1	1	G1′ · G2′ · G3 · G4′	SOK	1	0	0	0	0	1	0	0
S3	0	1	1	G1 + G2 + G4	SERR	1	0	1	0	0	1	0	0
S4	0	1	0	G1′ · G2′ · G3′ · G4′	S1	0	0	0	0	0	0	1	0
S4	0	1	0	G1′ · G2′ · G3′ · G4	SOK	1	0	0	0	0	0	1	0
S4	0	1	0	G1 + G2 + G3	SERR	1	0	1	0	0	0	1	0
SOK	1	0	0	G1 + G2 + G3 + G4	SOK	1	0	0	0	0	0	0	0
SOK	1	0	0	G1′ · G2′ · G3′ · G4′	S1	0	0	0	0	0	0	0	0
SERR	1	0	1	G1 + G2 + G3 + G4	SERR	1	0	1	0	0	0	0	1
SERR	1	0	1	G1′ · G2′ · G3′ · G4′	S1	0	0	0	0	0	0	0	1

*7.7.2 Estados sin utilizar

Nuestro diagrama de estado para el juego de las adivinanzas tiene seis estados, pero la máquina de estado real, construida a partir de tres flip-flops, tiene ocho. Al omitir los estados sin utilizar de la lista de transición, los tratamos como "sin importancia" en un sentido muy limitado:

- Cuando escribimos ecuaciones para Q1* y Q0*, formamos una suma de p términos de transición para combinaciones de estado/salida que tenían un 1 explícito en las columnas correspondientes de la lista de transición. Aunque no consideramos los estados sin utilizar, nuestro procedimiento implícitamente los trataba como si tuvieran ceros en las columnas Q1* y Q2*.
- A la inversa, escribimos la ecuación Q2*' como una suma de p-términos de transición para combinaciones de estado/entrada que tenían un 0 explícito en las columnas correspondientes de la lista de transición. Los estados sin utilizar eran tratados implícitamente como si tuvieran unos en la columna Q2*.

Como una consecuencia de estas elecciones, todos los estados, que no se utilizan en la máquina del juego de adivinanzas, tiene un estado siguiente codificado de 100 para todas las combinaciones de entrada. Éste es un comportamiento seguro y aceptable, en caso de que la máquina caiga en un estado sin utilizar, puesto que 100 es la codificación para uno de los estados normales (SOK).

Para tratar los estados sin utilizar como verdaderos "sin importancia", tendríamos que permitirles ir a cualquier estado siguiente bajo cualquier combinación de entrada. Esto es simple en principio, pero puede ser difícil en la práctica.

Al final de la sección 7.4.4 mostramos cómo manejar estados sin utilizar como "sin importancia" en el método del mapa de Karnaugh para desarrollar ecuaciones de transición/excitación. Desgraciadamente, para todos los problemas menos los pequeños, los mapas de Karnaugh son difíciles de manejar. Los programas de minimización lógica disponibles pueden manejar fácilmente problemas grandes, pero muchos de ellos no manejan elementos "sin importancia" o requieren que el diseñador inserte código especial para manejarlos. En las máquinas de estado ABEL, los estados siguientes sin importancia se pueden manejar utilizando la directiva `@DCSET`, como indicamos posteriormente en el cuadro de la página 632. En VHDL el proceso es un poco pesado.

*7.7.3 Asignación de estado codificado por salida

Examinemos otra realización de la máquina del juego de las adivinanzas. Las salidas de la máquina son una función del estado únicamente; además, se produce una combinación de salida *diferente* en cada estado nombrado. Por lo tanto, podemos emplear las salidas como variables de estado y asignar cada estado nombrado a la combinación de salida requerida. Esta clase de *asignación de estado codificado por salida* puede en ocasiones resultar en ecuaciones de excitación que son más simples que el conjunto de ecuaciones de excitación y salida que se obtiene una asignación de estado que utiliza un número mínimo de variables de estado.

asignación de estado codificado por salida

La tabla 7-19 es la lista de transición del juego de las adivinanzas que resulta de una asignación de estado codificado por salida. Cada ecuación de transición/excitación tiene muy pocos p términos de transición porque la lista de transición tiene muy pocos unos en las columnas de estado siguiente:

Tabla 7-19 Lista de transición para la máquina del juego de las adivinanzas que utiliza salidas como variables de estado.

Estado actual							*Estado siguiente*					
S	***L1***	***L2***	***L3***	***L4***	***ERR***	***Expresión de transición***	***S****	***L1****	***L2****	***L3****	***L4****	***ERR****
S1	1	0	0	0	0	G1′ · G2′ · G3′ · G4′	S2	0	1	0	0	0
S1	1	0	0	0	0	G1 · G2′ · G3′ · G4′	SOK	0	0	0	0	0
S1	1	0	0	0	0	G2 + G3 + G4	SERR	0	0	0	0	1
S2	0	1	0	0	0	G1′ · G2′ · G3′ · G4′	S3	0	0	1	0	0
S2	0	1	0	0	0	G1′ · G2 · G3′ · G4′	SOK	0	0	0	0	0
S2	0	1	0	0	0	G1 + G3 + G4	SERR	0	0	0	0	1
S3	0	0	1	0	0	G1′ · G2′ · G3′ · G4′	S4	0	0	0	1	0
S3	0	0	1	0	0	G1′ · G2′ · G3 · G4′	SOK	0	0	0	0	0
S3	0	0	1	0	0	G1 + G2 + G4	SERR	0	0	0	0	1
S4	0	0	0	1	0	G1′ · G2′ · G3′ · G4′	S1	1	0	0	0	0
S4	0	0	0	1	0	G1′ · G2′ · G3′ · G4	SOK	0	0	0	0	0
S4	0	0	0	1	0	G1 + G2 + G3	SERR	0	0	0	0	1
SOK	0	0	0	0	0	G1 + G2 + G3 + G4	SOK	0	0	0	0	0
SOK	0	0	0	0	0	G1′ · G2′ · G3′ · G4′	S1	1	0	0	0	0
SERR	0	0	0	0	1	G1 + G2 + G3 + G4	SERR	0	0	0	0	1
SERR	0	0	0	0	1	G1′ · G2′ · G3′ · G4′	S1	1	0	0	0	0

$$
\begin{aligned}
L1^* &= L1' \cdot L2' \cdot L3' \cdot L4 \cdot ERR' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + L1' \cdot L2' \cdot L3' \cdot L4' \cdot ERR' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
&\quad + L1' \cdot L2' \cdot L3' \cdot L4' \cdot ERR \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
L2^* &= L1 \cdot L2' \cdot L3' \cdot L4' \cdot ERR' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
L3^* &= L1' \cdot L2 \cdot L3' \cdot L4' \cdot ERR' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
L4^* &= L1' \cdot L2' \cdot L3 \cdot L4' \cdot ERR' \cdot (G1' \cdot G2' \cdot G3' \cdot G4') \\
ERR^* &= L1 \cdot L2' \cdot L3' \cdot L4' \cdot ERR' \cdot (G2 + G3 + G4) \\
&\quad + L1' \cdot L2 \cdot L3' \cdot L4' \cdot ERR' \cdot (G1 + G3 + G4) \\
&\quad + L1' \cdot L2' \cdot L3 \cdot L4' \cdot ERR' \cdot (G1 + G2 + G4) \\
&\quad + L1' \cdot L2' \cdot L3' \cdot L4 \cdot ERR' \cdot (G1 + G2 + G3) \\
&\quad + L1' \cdot L2' \cdot L3' \cdot L4' \cdot ERR \cdot (G1 + G2 + G3 + G4)
\end{aligned}
$$

No hay ecuaciones de salida, por supuesto. La ecuación ERR* anterior es la peor del grupo, requiriendo de 16 términos para expresarse ya sea en forma de suma de productos o producto de sumas mínima.

Como un grupo, las ecuaciones desarrolladas con anterioridad tienen apenas aproximadamente la misma complejidad que las ecuaciones de transición y de salida que desarrollamos de la tabla 7-18. Aun cuando la asignación codificada por salida no produce un conjunto más simple de ecuaciones en este ejemplo, todavía puede ahorrar costos en un diseño basado en PLD, puesto que en total se necesitan menos macroceldas o salidas PLD.

*7.7.4 Codificaciones de estado "sin importancia"

De los 32 posibles estados codificados que utilizan cinco variables, solamente seis se utilizan en la tabla 7-19. El resto de los estados no se utilizan y tienen un estado siguiente de 00000 si la máquina se construye con las ecuaciones en la subsección anterior. Otra posible disposición para estados sin utilizar, una que no hemos discutido antes, se obtiene mediante el uso cuidadoso de los elementos "sin importancia" en la asignación de estados codificados para los estados actuales.

La tabla 7-20 muestra una asignación de estado de esta clase para la máquina del juego de las adivinanzas, derivada de la asignación de estado codificada por salida de la subsección anterior. En este ejemplo, toda posible combinación de variables de estado actual corresponde a uno de los estados codificados (por ejemplo, 10111 = S1, 00101 = S3). Sin embargo, los *estados siguientes* son codificados empleando las mismas combinaciones únicas que en la subsección anterior. La tabla 7-21 muestra la lista de transición resultante.

En este enfoque, cada estado actual sin utilizar se comporta como un cercano estado "normal"; la figura 7-67 ilustra el concepto. La máquina se comporta bien y tiende a un "estado normal" si entra de manera involuntaria a un estado sin utilizar. Con todo el enfoque permite todavía alguna simplificación de la lógica de excitación y salida al introducir

Tabla 7-20
Asignación de estado actual para la máquina del juego de las adivinanzas, mediante el uso de elementos "sin importancia".

Estado	*L1*	*L2*	*L3*	*L4*	*ERR*
S1	1	x	x	x	x
S2	0	1	x	x	x
S3	0	0	1	x	x
S4	0	0	0	1	x
SOK	0	0	0	0	0
SERR	0	0	0	0	1

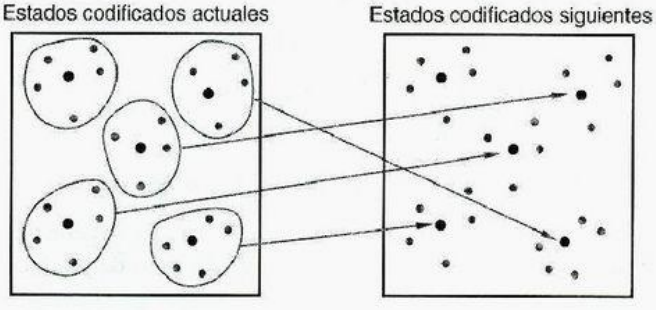


Figura 7-67
Asignación de estado empleando "sin importancia" para los estados actuales.

Tabla 7-21 Lista de transición para la máquina del juego de las adivinanzas mediante el uso de codificaciones de estado sin importancia.

Estado actual							*Estado siguiente*					
S	***L1***	***L2***	***L3***	***L4***	***ERR***	***Expresión de transición***	***S****	***L1****	***L2****	***L3****	***L4****	***ERR****
S	1	x	x	x	x	G1′ · G2′ · G3′ · G4′	S2	0	1	0	0	0
S	1	x	x	x	x	G1 · G2′ · G3′ · G4′	SOK	0	0	0	0	0
S	1	x	x	x	x	G2 + G3 + G4	SERR	0	0	0	0	1
S2	0	1	x	x	x	G1′ · G2′ · G3′ · G4′	S3	0	0	1	0	0
S2	0	1	x	x	x	G1′ · G2 · G3′ · G4′	SOK	0	0	0	0	0
S2	0	1	x	x	x	G1 + G3 + G4	SERR	0	0	0	0	1
S3	0	0	1	x	x	G1′ · G2′ · G3′ · G4′	S4	0	0	0	1	0
S3	0	0	1	x	x	G1′ · G2′ · G3 · G4′	SOK	0	0	0	0	0
S3	0	0	1	x	x	G1 + G2 + G4	SERR	0	0	0	0	1
S4	0	0	0	1	x	G1′ · G2′ · G3′ · G4′	S1	1	0	0	0	0
S4	0	0	0	1	x	G1′ · G2′ · G3′ · G4	SOK	0	0	0	0	0
S4	0	0	0	1	x	G1 + G2 + G3	SERR	0	0	0	0	1
SOK	0	0	0	0	0	G1 + G2 + G3 + G4	SOK	0	0	0	0	0
SOK	0	0	0	0	0	G1′ · G2′ · G3′ · G4′	S1	1	0	0	0	0
SERR	0	0	0	0	1	G1 + G2 + G3 + G4	SERR	0	0	0	0	1
SERR	0	0	0	0	1	G1′ · G2′ · G3′ · G4′	S1	1	0	0	0	0

elementos sin importancia en la lista de transición. Cuando se escribe un p término de transición de renglón, las variables de estado actual que no son sin importancia en ese renglón se omiten; por ejemplo,

$$\begin{aligned} ERR^* = {} & L1 \cdot (G2 + G3 + G4) \\ & + L1' \cdot L2 \cdot (G1 + G3 + G4) \\ & + L1' \cdot L2' \cdot L3 \cdot (G1 + G2 + G4) \\ & + L1' \cdot L2' \cdot L3' \cdot L4 \cdot (G1 + G2 + G3) \\ & + L1' \cdot L2' \cdot L3' \cdot L4' \cdot ERR \cdot (G1 + G2 + G3 + G4) \end{aligned}$$

En comparación con la ecuación de ERR* en la subsección anterior, la anterior ecuación aún requiere 16 términos para expresarse como una suma de productos. Sin embargo, requiere solamente cinco términos en la forma de producto de sumas mínima, lo que hace su complemento más adecuado para su realización en un PLD.

*7.8 Descomposición de las máquinas de estado

Del mismo modo que los procedimientos o subrutinas largos en un lenguaje de programación, las máquinas de estado extensas son difíciles de conceptualizar, diseñar y depurar. Por tanto, cuando nos enfrentamos a un problema extenso de máquina de estado, los diseñadores digitales con frecuencia buscan oportunidades para resolverlo con una colección de máquinas de estado más pequeñas.

descomposición de máquina de estado

Existe una teoría bien desarrollada de la *descomposición de máquina de estado* que se puede emplear para analizar cualquier máquina de estado monolítica dada a fin de determinar si puede ser realizada como una colección de otras más pequeñas. Sin embargo, la teoría de la descomposición no es demasiado útil para diseñadores que quieren evitar el diseño de extensas máquinas de estado en primer lugar. Más todavía, un diseñador práctico intenta proyectar el problema de diseño original en una estructura natural jerárquica, de modo que los usos y funciones de las submáquinas sean evidentes, haciendo innecesario escribir una tabla de estado para la máquina monolítica equivalente.

máquina principal
submáquinas

El tipo más simple y más comúnmente utilizado de descomposición se ilustra en la figura 7-68. Una *máquina principal* proporciona las entradas y salidas primarias y ejecuta el algoritmo de control de nivel superior. Las *submáquinas* efectúan los pasos de nivel inferior bajo el control de la máquina principal y pueden de manera opcional, manejar alguna de las entradas y salidas primarias.

Quizás la submáquina más comúnmente empleada sea un contador. La máquina principal inicia el contador cuando desea permanecer en un estado principal en particular para n tics de reloj, y el contador afirma una señal DONE ("hecho") cuando se han presentado los n tics. La máquina principal está diseñada para esperar en el mismo estado hasta que DONE es afirmada. Ésta agrega una entrada y salida extras a la máquina principal (START y DONE), pero ahorra $n - 1$ estados.

Un ejemplo de máquina de estado descompuesta diseñado a lo largo de estas líneas está basado en el juego de las adivinanzas de la sección 7.7. El juego de las adivinanzas original es fácil de ganar después de un minuto de práctica debido a que las lámparas siguen su ciclo a una velocidad muy consistente de 4 Hz. Para que el juego sea un desafío,

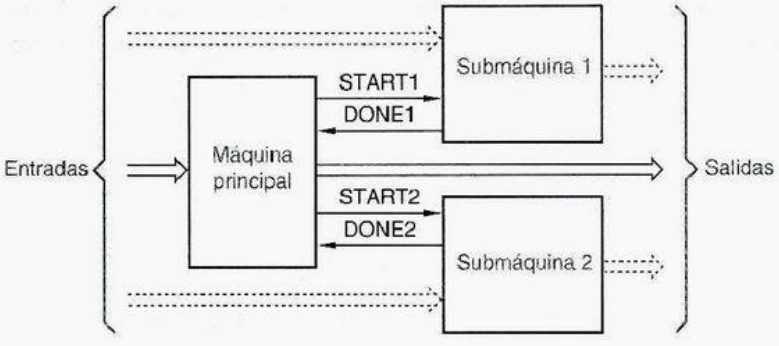


Figura 7-68
Una estructura de máquina de estado jerárquica, típica.

UNA BROMA REALMENTE MALA Advierta que el título de esta sección no tiene nada que ver con los flip-flops "enterrados" (integrados) encontrados en algunos PLD.

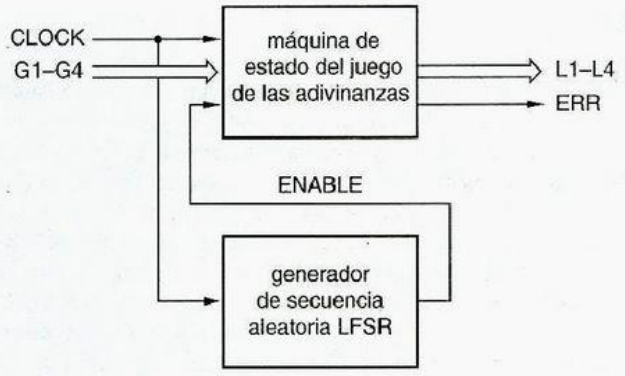


Figura 7-69
Diagrama de bloques del juego de las adivinanzas con retardo aleatorio.

podemos duplicar o triplicar la velocidad de reloj pero dejar que las lámparas permanezcan en cada estado durante un periodo aleatorio. Entonces el usuario en verdad debe adivinar si una lámpara dada permanecerá un tiempo suficientemente largo para que sea presionado el botón correspondiente.

Un diagrama de bloques para el juego de las adivinanzas mejorado se muestra en la figura 7-69. La máquina principal es básicamente la misma que antes, excepto en que avanza desde un estado de la lámpara hacia el siguiente solamente si la entrada de habilitación EN está afirmada, como se ilustra mediante el diagrama de estado en la figura 7-70. La entrada de habilitación es controlada por la salida de un generador de secuencia pseudoaleatoria, un registro de corrimiento de retroalimentación lineal (LFSR, "linear feedback shift register").

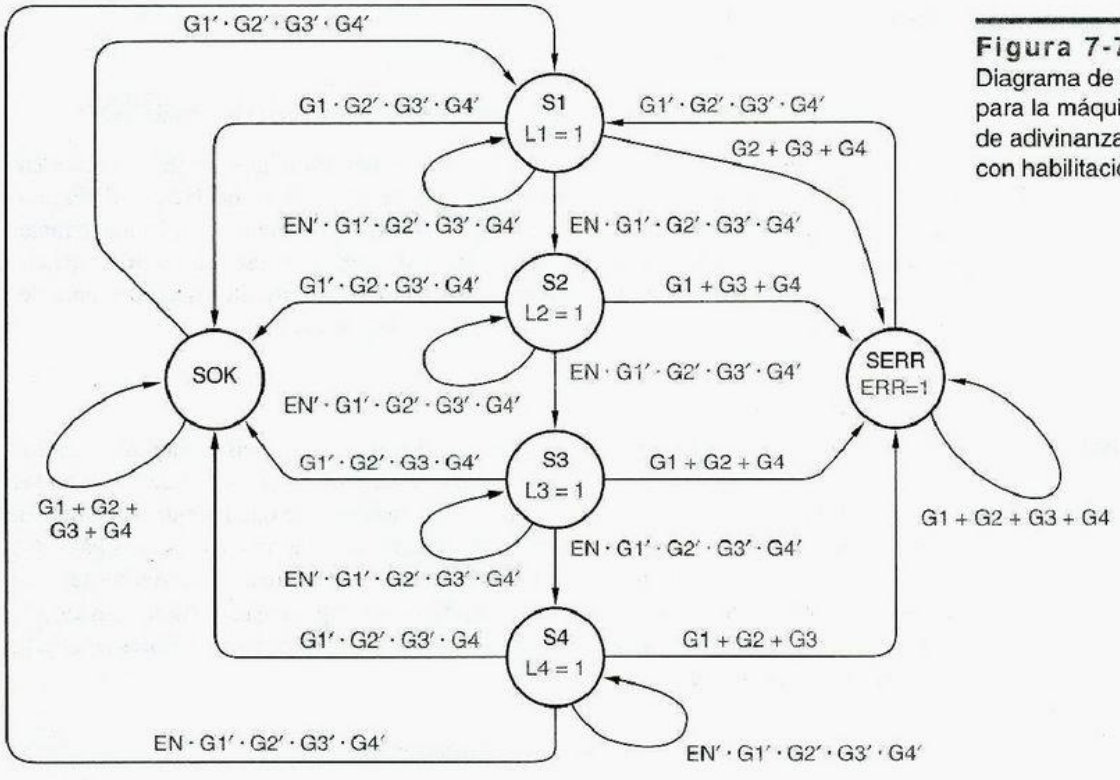


Figura 7-70
Diagrama de estado para la máquina de adivinanzas con habilitación.

UN CIRCUITO DESLIZANTE

Los circuitos LFSR se describen en la sección 8.5.8. En la figura 7-69, el bit de orden menor de un contador LFSR de n bits se utiliza como la señal de habilitación. De este modo, el tiempo que una lámpara permanece encendida depende de la secuencia de conteo del LFSR.

En el mejor de los casos para el usuario, el LFSR contiene 10...00; en este caso la lámpara se encuentra encendida durante $n - 1$ tics del reloj, porque toma ese intervalo de tiempo para el único 1 correrse a la posición del bit de menor orden. En el peor de los casos, el LFSR contiene 11...11, y el corrimiento ocurre para n tics de reloj consecutivos. En otros momentos, el corrimiento se detiene por un tiempo determinado por el número de ceros consecutivos comenzando con el bit de menor orden del LFSR.

Todos estos casos son bastante impredecibles, a menos que el usuario haya memorizado el ciclo de corrimiento ($2^n - 1$ estados) o sea muy rápido para la aritmética de campo de Galois. Obviamente, un valor grande de n (≥ 16) proporciona la mayor diversión.

Otro candidato evidente para la descomposición es una máquina de estado que realice multiplicación binaria mediante el algoritmo de corrimiento y suma, o de división binaria empleando el algoritmo de corrimiento y resta. Para realizar una operación de n bits, estos algoritmos requieren de un paso de inicialización, n pasos de cálculo y posibles pasos de limpieza. La máquina principal para un algoritmo de esta naturaleza contiene estados para inicialización, cálculo genérico y pasos de limpieza y puede utilizarse un contador módulo n como una submáquina para controlar el número de pasos de cálculo genérico ejecutados.

*7.9 Circuitos secuenciales de retroalimentación

El biestable simple y los diversos latchs y flip-flops que estudiamos anteriormente en este capítulo son en su totalidad circuitos secuenciales de retroalimentación. Cada uno tiene uno o más lazos de retroalimentación que, ignorando sus comportamientos durante las transiciones de estado, almacenan un 0 o un 1 siempre. Los lazos de retroalimentación son elementos de memoria, y el comportamiento de los circuitos depende tanto de las entradas actuales como de los valores almacenados en los lazos.

*7.9.1 Análisis

circuitos en modo fundamental

Los circuitos secuenciales de retroalimentación son el ejemplo más común de *circuitos en modo fundamental*. En tales circuitos, normalmente no se permite que las entradas cambien simultáneamente. El procedimiento de análisis supone que las entradas cambian una a la vez, permitiendo que haya suficiente tiempo entre cambios sucesivos para que el circuito se establezca en un estado interno estable. Esto difiere de los circuitos temporizados, en los que varias entradas pueden cambiar en tiempos casi arbitrarios sin afectar el estado, se realiza un muestreo de todos los valores de entrada, y los cambios de estado ocurren con respecto a una señal de reloj.

* Esta sección y todas sus subsecciones son opcionales.

CONSERVE SU RETROALIMENTACIÓN PARA USTED MISMO

Sólo rara vez un diseñador lógico encuentra una situación donde un circuito secuencial de retroalimentación debe ser analizado o diseñado. Los circuitos secuenciales de retroalimentación más comúnmente utilizados son los flip-flops y los latchs que son empleados como los bloques de construcción en diseños de circuito secuencial más grandes. Su diseño interno y sus especificaciones de funcionamiento se suministran por un fabricante de CI.

Incluso un diseño ASIC por lo regular no diseña circuitos de latch o flip-flop al nivel de compuerta, puesto que estos elementos son suministrados en una "biblioteca" de funciones comúnmente empleadas en la tecnología ASIC particular. Con todo, usted puede tener curiosidad acerca de cómo los flip-flops y latchs comerciales "hacen su trabajo"; esta sección le mostrará cómo analizar tales circuitos.

Como las máquinas de estado sincrónicas temporizadas, los circuitos secuenciales de retroalimentación pueden estructurarse como circuitos de Mealy o de Moore, como se ilustra en la figura 7-71. Un circuito con n lazos de retroalimentación tienen n variables de estado binarias y 2^n estados.

Para analizar un circuito secuencial de retroalimentación, debemos dividir los lazos de retroalimentación en la figura 7-71 de manera que el valor siguiente almacenado en cada lazo pueda predecirse como una función de las entradas del circuito y el valor actual almacenado en todos los lazos. La figura 7-72 muestra cómo hacer esto para el circuito NAND para una celda D, el cual tiene solamente un lazo de retroalimentación. Conceptualmente dividimos el lazo insertando un buffer ficticio en el lazo como se muestra a continuación. La salida del buffer, denominada Y, es la variable de estado simple para este ejemplo.

Supongamos que el retardo de propagación del buffer ficticio es de 10 ns (pero cualquier número distinto de cero bastará) y que todos los otros componentes del circuito tienen retardo cero. Si conocemos el estado actual del circuito (Y) y las entradas (D y C), entonces podemos predecir el valor que Y tendrá en 10 ns. El valor siguiente de Y, denotado Y*, es una función combinacional de las entradas y estado actuales. De esta manera, leyendo el diagrama de circuito, podemos escribir una *ecuación de excitación para* Y*.

ecuación de excitación

$$\begin{aligned} Y^* &= (C \cdot D) + (C \cdot D' + Y')' \\ &= C \cdot D + C' \cdot Y + D \cdot Y \end{aligned}$$

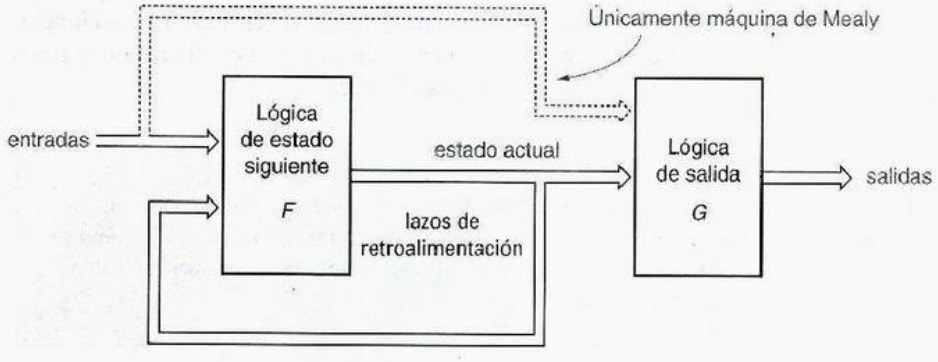


Figura 7-71
Estructura de circuito secuencial de retroalimentación para máquinas de Mealy y de Moore.

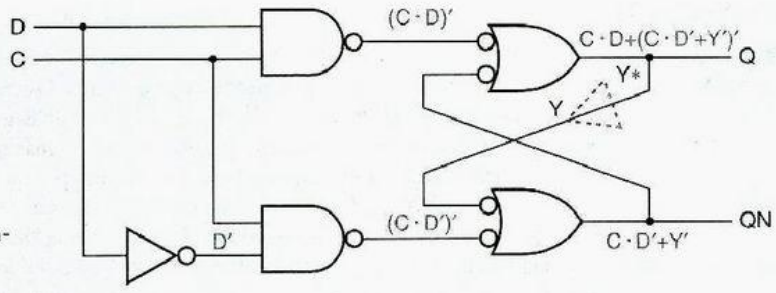


Figura 7-72
Análisis de retroalimentación de un latch D.

tabla de transición

Ahora el estado del lazo de retroalimentación (y el circuito) pueden escribirse como una función de la entrada y estado actuales, y numerado por una *tabla de transición* como se ilustra en la figura 7-73. Cada celda en la tabla de transición muestra el valor de salida del buffer ficticio que se presentará 10 ns (o cualquier retardo que se haya supuesto) después de que ocurra la combinación de entrada y estado correspondiente.

Figura 7-73
Tabla de transición para el cerrojo D de la figura 7-72.

	C D			
Y	00	01	11	10
0	0	0	1	0
1	1	1	1	0
	Y*			

Una tabla de transición tiene un renglón para cada posible combinación de las variables de estado, de modo que un circuito con n lazos de retroalimentación tiene 2^n renglones en su tabla de transición. La tabla tiene una columna para cada posible combinación de entrada, de manera que un circuito con m entradas tiene 2^m columnas en su tabla de transición.

Por definición, un circuito en modo fundamental tal como un circuito secuencial de retroalimentación no tiene un reloj que indique cuándo debe muestrear sus entradas. En vez de ello, podemos imaginar que el circuito está evaluando su propio estado y entrada actuales *continuamente* (o cada 10 ns, si se prefiere). Como resultado de cada evaluación, se va hacia un estado siguiente predicho por la tabla de transición. La mayor parte del tiempo, el estado siguiente es el mismo que el estado actual; esto es la esencia del funcionamiento en modo fundamental. Haremos algunas definiciones a continuación que nos ayudarán a estudiar este comportamiento con más detalle.

SÓLO UN LAZO La manera en que el circuito de la figura 7-72 es dibujado, puede hacer parecer como que hay dos lazos de retroalimentación. Sin embargo, una vez que efectuamos una división como se muestra, no hay más lazos. Es decir, cada señal puede escribirse como una función combinacional de las otras señales, sin incluirse a sí misma.

	C D			
S	00	01	11	10
S0	(S0)	(S0)	S1	(S0)
S1	(S1)	(S1)	(S1)	S0
	S*			

Figura 7-74
Tabla de estado para el latch D en la figura 7-72, mostrando los estados totales estables.

En un circuito en modo fundamental, un *estado total* es una combinación particular de *estado interno* (los valores almacenados en los lazos de retroalimentación) y el *estado de entrada* (el valor actual de las entradas del circuito). Un *estado total estable* es una combinación de estado interno y estado de entrada de tal forma que el estado interno siguiente predicho por la tabla de transición es el mismo que el estado interno actual. Si el estado interno siguiente es diferente, entonces la combinación es un *estado total inestable*. Hemos vuelto a escribir la tabla de transición para el latch D en la figura 7-74 como una *tabla de estado*, dando los nombres S0 y S1 a los estados y dibujando un círculo alrededor de los estados totales estables.

estado total
estado interno
estado de entrada
estado total estable
estado total inestable
tabla de estado

Para completar el análisis del circuito, también debemos determinar cómo se comportan las salidas como funciones de las entradas y estado interno. Existen dos salidas y por tanto dos *ecuaciones de salida*:

ecuación de salida

$$Q = C \cdot D + C' \cdot Y + D \cdot Y$$
$$QN = C \cdot D' + Y'$$

Nótese que Q y QN son *salidas*, no variables de estado. Aun cuando el circuito tiene dos salidas, lo que teóricamente toma cuatro combinaciones, tiene solamente una variable de estado Y, y por tanto únicamente dos estados.

Los valores de salida que predicen las ecuaciones de Q y QN pueden incorporarse en una tabla de salida y estado combinados que describen completamente la operación del circuito, como se ilustra en la figura 7-75. Aunque Q y QN son normalmente complementarias, es posible para ellas tener el mismo valor (1) momentáneamente, durante la transición de S0 a S1 bajo la columna C D = 11 de la tabla.

Ahora podemos predecir el comportamiento del circuito desde la tabla de salida y transición. Antes que nada, note que hemos escrito las etiquetas de columna en nuestras tablas de estado en orden de "mapa de Karnaugh", de modo que sólo cambie un bit de entrada entre las columnas adyacentes de la tabla. Esta disposición ayuda a nuestro análisis

	C D			
S	00	01	11	10
S0	(S0), 01	(S0), 01	S1, 11	(S0), 01
S1	(S1), 10	(S1), 10	(S1), 10	S0, 01
	S*, Q QN			

Figura 7-75
Tabla de estado y salida para el cerrojo D.

	C D			
S	00	01	11	10
S0	(S0), 01	(S0), 01	S1 , 11	(S0), 01
S1	(S1), 10	(S1), 10	(S1), 10	S0 , 01
	S*, Q QN			

Figura 7-76
Análisis del latch D para unas cuantas transiciones.

porque suponemos que solamente una entrada cambia a un tiempo y que el circuito siempre alcanza un estado estable antes de que otra entrada cambie.

En cualquier momento, el circuito se encuentra en un estado interno particular y una entrada particular se le aplica; llamamos a esta combinación el estado total del circuito. Comencemos en el estado total estable "S0/00" (S = S0, C D = 00), como se muestra en la figura 7-76. Ahora supongamos que cambiamos D a 1. El estado total se mueve a una celda a la derecha; tenemos un nuevo estado total estable, S0/01. La entrada D es diferente, pero la salida y el estado interno son los mismos como antes. A continuación, cambiemos C a 1. El estado total se mueve a una celda a la derecha para S0/11, que es inestable. La entrada del siguiente estado en esta celda envía el circuito al estado interno S1, de modo que el estado total se mueve hacia abajo una celda, S1/11. Examinando la entrada del siguiente estado en la nueva celda, hallamos que hemos alcanzado un estado total estable. Podemos trazar el comportamiento del circuito para cualquier secuencia deseada de cambios de entrada simple de esta manera.

Ahora, podemos reconsiderar la pregunta de los cambios de entrada simultáneos. Aun cuando cambios de entrada "casi simultáneos" pueden presentarse en la práctica, debemos suponer que nada ocurre de manera simultánea a fin de analizar el comportamiento de los circuitos secuenciales. La imposibilidad de los eventos simultáneos está apoyada por los retardos variantes de los mismos componentes del circuito, los cuales dependen de parámetros de fabricación, voltaje y temperatura. Esto significa que un conjunto de n entradas que al parecer cambian en forma "simultánea", en realidad puede cambiar en cualquiera de $n!$ diferentes órdenes desde el punto de vista del funcionamiento del circuito.

Por ejemplo, considere el funcionamiento del latch D como se muestra en la figura 7-77. Supongamos que inicia en el estado total estable S1/11. Ahora supongamos que C y D son "simultáneamente" establecidos a 0. En realidad, el circuito se comporta como si

	C D			
S	00	01	11	10
S0	(S0), 01	(S0), 01	S1 , 11	(S0), 01
S1	(S1), 10	(S1), 10	(S1), 10	S0 , 01
	S*, Q QN			

Figura 7-77
Múltiples cambios de entrada con el latch D.

una u otra entrada llegara primero a 0. Supongamos que C cambia primero. Entonces la secuencia de dos flechas apuntando a la izquierda en la tabla nos dice que el circuito va al estado total estable S1/00. Sin embargo, si D cambia primero, entonces la otra secuencia de flechas nos dice que el circuito tiende al estado total estable S0/00. Así el estado final del circuito es impredecible, una pista de que el ciclo de retroalimentación puede en realidad llegar a ser metaestable si establecemos C y D a 0 de manera simultánea. El lapso en el cual esta visión de simultaneidad es relevante es la ventana del tiempo de establecimiento y de retención del latch D.

Los cambios de entrada simultáneos no siempre causan un comportamiento impredecible. Sin embargo, debemos analizar los efectos de todas las posibles ordenaciones de cambios de señal para determinar esto; si todas las ordenaciones dan el mismo resultado, entonces la salida del circuito es predecible. Por ejemplo, considere el comportamiento del latch D comenzando en el estado total S0/00 con C y D cambiando simultáneamente de 0 a 1; siempre acaba en el estado total S1/11.

*7.9.2 Análisis de circuitos con lazos de retroalimentación múltiple

En circuitos con múltiples lazos de retroalimentación, debemos dividir todos los lazos, creando un buffer ficticio y variable de estado para cada lazo que dividamos. Existen muchas maneras posibles, que los matemáticos llaman *conjuntos de corte*, para dividir los lazos en un circuito dado, de manera que, ¿cómo sabemos cuál es el mejor? La respuesta es que cualquier *conjunto de corte mínimo* (un conjunto de corte con un número mínimo de cortes) es adecuado. Los matemáticos pueden ofrecer un algoritmo para encontrar un conjunto de corte mínimo, pero como un diseñador digital trabaja en circuitos pequeños, se puede únicamente observar el circuito para hallar uno.

conjunto de corte

conjunto de corte mínimo

Diferentes conjuntos de corte para un circuito conducen a diferentes ecuaciones de excitación, tablas de transición y tablas de estado/salida. Sin embargo, los estados totales estables derivados de un conjunto de corte mínimo corresponden de manera unívoca a los estados totales estables derivados de cualquier otro conjunto de corte mínimo para el mismo circuito. Es decir, las tablas de estado/salida derivadas de diferentes conjuntos de corte mínimo exhiben el mismo comportamiento de entrada/salida con sólo los nombres y codificación de los estados cambiados.

Si se utiliza más del número mínimo de cortes para analizar un circuito secuencial de retroalimentación, la tabla resultante de estado/salida aún describirá el circuito correctamente. Sin embargo, utilizaremos 2^m veces tantos estados como sea necesario, donde m es el número de cortes extra. Los procedimientos de minimización de estado formal se pueden utilizar para reducir esta tabla al tamaño adecuado, pero es preferible la idea de seleccionar un conjunto de corte mínimo en primer lugar.

Un buen ejemplo de circuito secuencial con múltiples lazos de retroalimentación es el diseño de circuito comercial para un flip-flop D TTL disparado por flanco positivo que mostramos en la figura 7-20. El circuito se ha redibujado en forma simplificada en la figura 7-78, suponiendo que las entradas PR_L y CLR_L del circuito original nunca son afirmadas, y también muestran buffers ficticios para dividir los tres lazos de retroalimentación. Estos tres lazos dan origen a ocho estados, en comparación con el mínimo de cuatro estados empleados por el diseño de dos lazos en la figura 7-19. Nos dirigiremos a esta curiosa diferencia posteriormente.

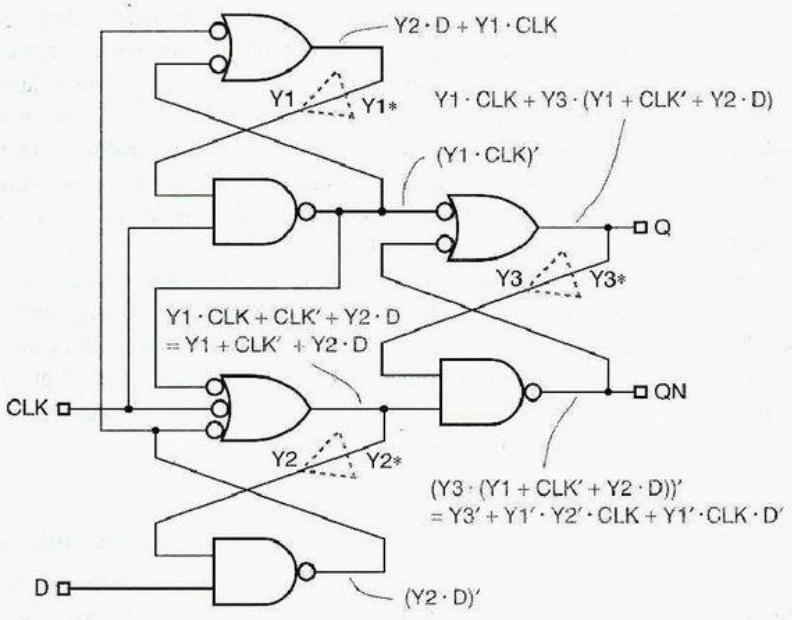


Figura 7-78
Flip-flop D disparado por flanco positivo simplificado para análisis.

Las siguientes ecuaciones de excitación y salida pueden obtenerse a partir del diagrama lógico en la figura 7-78:

$$Y1* = Y2 \cdot D + Y1 \cdot CLK$$
$$Y2* = Y1 + CLK' + Y2 \cdot D$$
$$Y3* = Y1 \cdot CLK + Y1 \cdot Y3 + Y3 \cdot CLK' + Y2 \cdot Y3 \cdot D$$
$$Q = Y1 \cdot CLK + Y1 \cdot Y3 + Y3 \cdot CLK' + Y2 \cdot Y3 \cdot D$$
$$QN = Y3' + Y1' \cdot Y2' \cdot CLK + Y1' \cdot CLK \cdot D'$$

Figura 7-79
Tabla de transición para el flip-flop D de la figura 7-78.

	CLK D			
Y1 Y2 Y3	00	01	11	10
000	010	010	(000)	(000)
001	011	011	000	000
010	(010)	110	110	000
011	(011)	111	111	000
100	010	010	111	111
101	011	011	111	111
110	010	(110)	111	111
111	011	(111)	(111)	(111)
	Y1* Y2* Y3*			

	CLK D			
Y1 Y2 Y3	00	01	11	10
000	010	010	(000)	(000)
001	011	011	000	000
010	(010)	110	110	000
011	(011)	111	111	000
	Y1* Y2* Y3*			

Figura 7-80
Segmento de la tabla de transición del flip-flop D mostrando una carrera no crítica.

La tabla de transición correspondiente se muestra en la figura 7-79, con los estados totales estables encerrados en círculos. Antes de ir más adelante, debemos introducir el concepto de "carreras".

*7.9.3 Carreras

carrera

En un circuito secuencial de retroalimentación, se dice que una *carrera* se presenta cuando múltiples variables internas cambian de estado como un resultado del cambio de estado de una sola entrada. En el ejemplo de la figura 7-79, se presenta una carrera en estado total estable 011/00 cuando CLK se cambia de 0 a 1. La tabla indica que el estado interno siguiente es 000, un cambio de 2 variables de 011.

Como lo discutimos con anterioridad, las señales lógicas en realidad nunca cambian "simultáneamente". De este modo, el estado interno puede hacer el cambio 011 → 000 como cualquiera de éstos: 011 → 001 → 000 o 011 → 010 → 000. La figura 7-80 indica que el circuito del ejemplo, comenzando en el estado total 011/00, debería ir al estado total 000/10 cuando CLK cambie de 0 a 1. Sin embargo, puede visitar temporalmente el estado total 001/10 o 010/010 en el trayecto. Esto está bien, porque el estado interno siguiente para ambos estados temporales es 000; por consiguiente, incluso en los estados temporales, la lógica de excitación continúa dirigiendo los lazos de retroalimentación hacia el mismo estado total estable, 000/10. Puesto que el estado final no depende del orden en el que cambian las variables de estado, esto se conoce como una *carrera no crítica*.

carrera no crítica

Ahora supongamos que la entrada de estado siguiente para el estado total 010/10 se cambia a 110, como se ilustra en la figura 7-81, y considérese el caso que acabamos de analizar. Comenzando en el estado total estable 011/00 y cambiando CLK a 1, el circuito puede terminar en el estado interno 000 o 111, dependiendo del orden y la rapidez de los cambios de variable internos. Esto se conoce como una *carrera crítica*.

carrera crítica

¡ESTÉ PENDIENTE DE LAS CARRERAS CRÍTICAS!

Cuando usted diseñe un circuito secuencial basado en retroalimentación, debe estar seguro de que su tabla de transición no contenga ninguna carrera crítica. De otro modo, el circuito puede funcionar de manera impredecible, con el estado siguiente para transiciones críticas dependiendo de factores como temperatura, voltaje y las fases de la luna.

	CLK D			
Y1 Y2 Y3	00	01	11	10
000	010	010	(000)	(000)
001	011	011	000	000
010	(010)	110	110	110
011	(011)	111	111	000
100	010	010	111	111
101	011	011	111	111
110	010	(110)	111	111
111	011	(111)	(111)	(111)
	Y1* Y2* Y3*			

Figura 7-81
Una tabla de transición conteniendo una carrera crítica.

*7.9.4 Tablas de estado y tablas de flujo

El análisis de la tabla de transición real, figura 7-79, para nuestro ejemplo del circuito del flip-flop D, muestra que no tiene ninguna carrera crítica; de hecho, no tiene carreras excepto la carrera no crítica que identificamos anteriormente. Una vez que hemos determinado este hecho, ya no necesitamos hacer referencia a variables de estado. En su lugar, podemos nombrar las combinaciones estado-variable y determinar los valores de salida para cada combinación estado/entrada para obtener una tabla estado/salida como la de la figura 7-82.

Figura 7-82
Tabla de estado/salida para el flanco D en la figura 7-78.

	CLK D			
S	00	01	11	10
S0	S2 , 01	S2 , 01	(S0) , 01	(S0) , 01
S1	S3 , 10	S3 , 10	S0 , 01	S0 , 10
S2	(S2) , 01	S6 , 01	S6 , 01	S0 , 01
S3	(S3) , 10	S7 , 10	S7 , 10	S0 , 01
S4	S2 , 01	S2 , 01	S7 , 11	S7 , 11
S5	S3 , 10	S3 , 10	S7 , 10	S7 , 10
S6	S2 , 01	(S6) , 01	S7 , 11	S7 , 11
S7	S3 , 10	(S7) , 10	(S7) , 10	(S7) , 10
	S* , Q QN			

	CLK D			
S	00	01	11	10
S0	S2 , 01	S6 , 01	(S0) , 01	(S0) , 01
S2	(S2) , 01	S6 , 01	— , –	S0 , 10
S3	(S3) , 10	S7 , 10	— , –	S0 , 01
S6	S2 , 01	(S6) , 01	S7 , 11	— , –
S7	S3 , 10	(S7) , 10	(S7) , 10	(S7) , 10
	S* , Q QN			

Figura 7-83
Tabla de salida y flujo para el flip-flop D en la figura 7-78.

La tabla de estado muestra lo siguiente: para algunos cambios de una sola entrada, el circuito toma múltiples "saltos" para obtener un nuevo estado total y estable. Por ejemplo, en el estado S0/11, un cambio de entrada a 01 envía al circuito primero al estado S2 y después al estado total estable S6/01. Una *tabla de flujo* elimina los saltos múltiples y muestra solamente el destino final para cada transición. La tabla de flujo también elimina los renglones que corresponden a los estados internos que no se utilizan, aquellos que son estables para ninguna combinación de entrada, y elimina las entradas de estado-siguiente para los estados totales que no se pueden obtener desde un estado total estable, como resultado de un único cambio de entrada. Al aplicar estas reglas, la figura 7-83 muestra la tabla de flujo para nuestro ejemplo del flip-flop D.

tabla de flujo

El comportamiento disparado por flanco del flip-flop puede observarse en la serie de transiciones de estado que se indican en la figura 7-84. Supongamos que el flip-flop comienza en el estado interno S0/10. Es decir, el flip-flop está almacenando un 0 (puesto que Q = 0), CLK es 1, y D es 0. Ahora supongamos que D cambia a 1; la tabla de flujo muestra que movemos una celda a la izquierda, todavía un estado total estable con el mismo valor de salida. Podemos cambiar D entre 0 y 1 tanto como queramos, y sólo rebotar hacia atrás y hacia adelante entre estas dos celdas. Sin embargo, una vez que CLK cambia

	CLK D			
S	00	01	11	10
S0	S2 , 01	S6 , 01	(S0) , 01	(S0) , 01
S2	(S2) , 01	S6 , 01	— , –	S0 , 10
S3	(S3) , 10	S7 , 10	— , –	S0 , 01
S6	S2 , 01	(S6) , 01	S7 , 11	— , –
S7	S3 , 10	(S7) , 10	(S7) , 10	(S7) , 10
	S* , Q QN			

Figura 7-84
Tabla de salida y flujo mostrando el comportamiento disparado por flanco de los flip-flops D.

	CLK D			
S	00	01	11	10
SB	(SB) , 01	S6 , 01	(SB) , 01	(SB) , 01
S3	(S3) , 10	S7 , 10	— , –	SB , 01
S6	SB , 01	(S6) , 01	S7 , 11	— , –
S7	S3 , 10	(S7) , 10	(S7) , 10	(S7) , 10
	S* , Q QN			

Figura 7-85
Tabla de salida y flujo reducida para un flip-flop D disparado por flanco positivo.

a 0, nos movemos al estado interno S2 o S6, dependiendo de si D era 0 o 1 en el momento; pero aún la salida se encuentra sin cambio. Una vez más, podemos cambiar D entre 0 y 1 tanto como queramos, esta vez oscilando entre S2 y S6 sin cambiar la salida.

El momento de la verdad viene finalmente cuando CLK cambia a 1. Dependiendo si estamos en S2 o en S6, regresamos a S0 (dejando Q en 0) o a S7 (estableciendo Q a 1). Un comportamiento similar que involucra a S3 y S7 puede observarse en un flanco de reloj positivo (ascendente) que provoca que Q cambie de 1 a 0.

En la figura 7-19 mostramos un circuito para un flip-flop D disparado por flanco positivo con solamente dos lazos de retroalimentación y por lo tanto cuatro estados. El circuito que acabamos de analizar tiene tres lazos y ocho estados. Aun después de eliminar los estados sin utilizar, la tabla de flujo tiene cinco estados. Sin embargo, un procedimiento de minimización de estado formal puede emplearse para mostrar que los estados S0 y S2 son "compatibles", de manera que se pueden mezclar en un solo estado SB que maneja las transiciones para ambos estados originales, como se ilustra en la figura 7-85. De este modo, un circuito de cuatro estados podría realizar este trabajo. En el ejercicio 7.66 usted demostrará que el circuito en la figura 7-19 hace el trabajo especificado por la tabla de flujo reducida.

*7.9.5 Análisis de flip-flop D CMOS

Los flip-flops CMOS por lo regular emplean compuertas de transmisión en sus lazos de retroalimentación. Por ejemplo, la figura 7-86 muestra el diseño del circuito del flip-flop D disparado por flanco positivo "FD1Q" en la serie LCA500K de LSI Logic de arreglos de compuerta CMOS. Un flip-flop de esta clase puede ser analizado del mismo modo que un diseño puramente basado en compuerta lógica, una vez que se reconocen los lazos de retroalimentación. La figura 7-86 tiene dos lazos de retroalimentación, cada uno de los cuales tiene un par de compuertas de transmisión en una configuración tipo multiplexor controlada por CLK y CLK', produciendo las siguientes ecuaciones de lazo:

$$Y1* = CLK' \cdot D' + CLK \cdot Y1$$
$$Y2* = CLK \cdot Y1' + CLK' \cdot Y2$$

Excepto por la doble inversión de los datos a medida que pasan de D a Y2* (una vez en la ecuación Y1* y de nueva cuenta en la ecuación Y2*), estas ecuaciones nos recuerdan

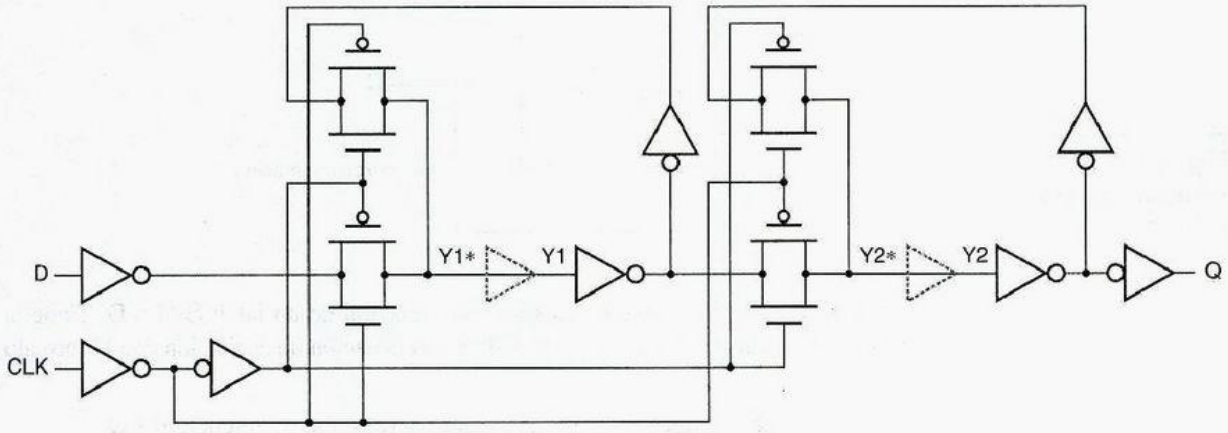


Figura 7-86 Flip-flop D CMOS disparado por flanco positivo para análisis.

mucho la estructura del latch maestro/esclavo del flip-flop D en la figura 7-15. Se deja como un ejercicio (7.73) el completar el análisis formal del circuito. Advierta, sin embargo, que ya que existen apenas dos lazos de retroalimentación, el estado resultante y tablas del flujo tendrán el mínimo de precisamente cuatro estados.

DISEÑO DE CIRCUITO DE RETROALIMENTACIÓN

Los circuitos secuenciales de retroalimentación que hemos analizado en esta sección exhiben un comportamiento bastante razonable, puesto que, después de todo, son circuitos flip-flop y de latch que han sido utilizados por años. Sin embargo, si juntamos una colección "aleatoria" de compuertas y lazos de retroalimentación, no necesariamente obtendremos un comportamiento de circuito secuencial "razonable". En unos ciertos casos poco comunes no obtendremos un circuito secuencial (véase el ejercicio 7.67), y en muchos casos el circuito puede ser inestable para alguna o la totalidad de las combinaciones de entrada (véase el ejercicio 7.71). De este modo, el diseño de circuitos secuenciales de retroalimentación continúa siendo algo como de magia negra y es practicado solamente por una pequeña fracción de los diseñadores digitales. Con todo, la sección siguiente presenta conceptos básicos que le ayudarán a hacer diseños simples.

*7.10 Diseño de circuito secuencial de retroalimentación

En ocasiones es útil diseñar un pequeño circuito secuencial de retroalimentación, tal como un latch especializado o un receptor de pulsos; esta sección le mostrará cómo. Es incluso posible que usted pueda llegar a ser un diseñador de CI y ser responsable de diseñar flip-flops y latchs de alto rendimiento de la nada. Esta sección le servirá como una introducción a los conceptos básicos que usted necesitará, pero todavía necesitará una cantidad considerable de estudio, experiencia y sutileza para hacerlo bien.

*7.10.1 Latchs

Aunque el diseño de circuitos secuenciales de retroalimentación es generalmente un problema difícil, algunos circuitos se pueden diseñar con bastante facilidad. Cualquier circuito

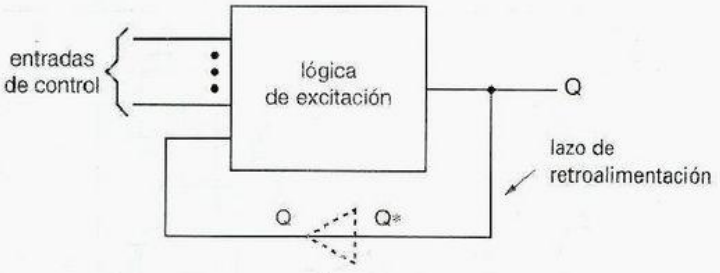


Figura 7-87
Estructura general de un latch.

con un lazo de retroalimentación es sólo una variación de un latch S-R o D. Tiene la estructura general mostrada en la figura 7-87 y una ecuación de excitación con el formato siguiente:

$$Q^* = (\text{término de forzamiento}) + (\text{término de retención}) \cdot Q$$

Por ejemplo, las ecuaciones de excitación para los latchs S-R y D son

$$Q^* = S + R' \cdot Q$$

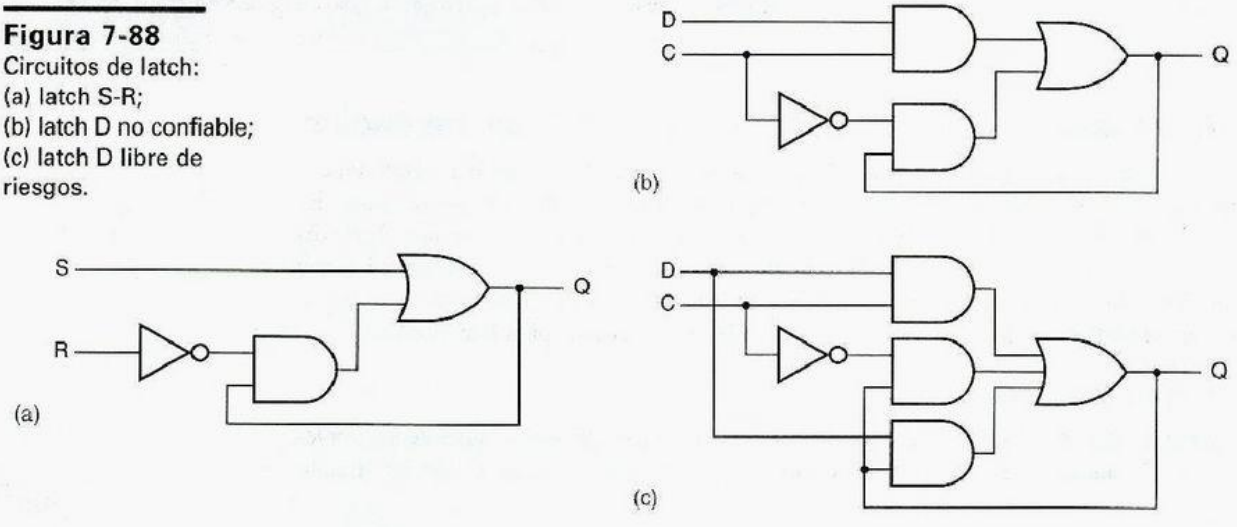
$$Q^* = C \cdot D + C' \cdot Q$$

Los circuitos correspondientes se ilustran en la figura 7-88(a) y (b).

lógica de excitación libre de riesgos

En general, *la lógica de excitación en un circuito secuencial de retroalimentación debe estar libre de riesgos*; demostraremos este hecho por medio de un ejemplo. La figura 7-89(a) es un mapa de Karnaugh para el circuito de excitación del latch D de la figura 7-88(b). El mapa exhibe un riesgo estático 1 cuando D y Q son 1, y C está cambiando. Por desgracia, el lazo de retroalimentación del latch no puede retener su valor si se presenta una interferencia o "glitch" en consecuencia del riesgo. Por ejemplo, considere lo que ocurre cuando D y Q son 1 y C cambia de 1 a 0; el circuito debería asegurar un 1. Sin embargo, a menos que el inversor sea muy rápido, la salida de la compuerta AND superior se va a 0 antes de que la salida de la inferior se vaya a 1, la salida de a compuerta OR vaya a 0 y el lazo de retroalimentación almacene un 0.

Figura 7-88
Circuitos de latch:
(a) latch S-R;
(b) latch D no confiable;
(c) latch D libre de riesgos.



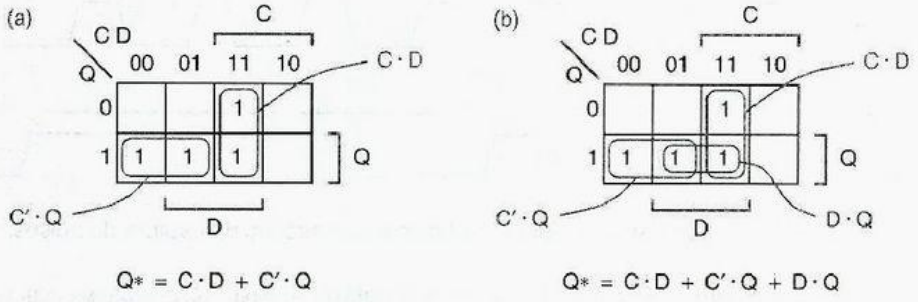


Figura 7-89
Mapas de Karnaugh para funciones de excitación de latch D: (a) original, conteniendo un riesgo estático 1; (b) riesgo eliminado.

Los riesgos se pueden eliminar utilizando los métodos descritos en la sección 4.5. En el latch D, simplemente incluimos el término de consenso en la ecuación de excitación:

$$Q^* = C \cdot D + C' \cdot Q + D \cdot Q$$

La figura 7-88(c) muestra el correspondiente circuito de latch D correcto, libre de riesgos.

Ahora, suponga que necesitamos un latch "D" especializado con tres entradas de datos, D1-D3, que almacene un 1 solamente si D1-D3 = 010. Podemos convertir esta descripción en palabras a una ecuación de excitación que mimetice la ecuación para un latch D simple:

$$Q^* = C \cdot (D1' \cdot D2 \cdot D3') + C' \cdot Q$$

Eliminando riesgos, obtenemos

$$Q^* = C \cdot D1' \cdot D2 \cdot D3' + C' \cdot Q + D1' \cdot D2 \cdot D3' \cdot Q$$

la ecuación de excitación libre de riesgos puede realizarse con compuertas discretas o en un PLD, como lo mostramos en la sección 8.2.6.

*7.10.2 Diseño de tabla de flujo de modo fundamental

Para diseñar circuitos secuenciales de retroalimentación más complejos de los latchs, primero debemos convertir la descripción en palabras a una tabla de flujo. Una vez que tengamos una tabla de flujo, podemos poner manos a la obra (con algún esfuerzo) para obtener un circuito.

Cuando construimos la tabla de flujo para un circuito secuencial de retroalimentación, proporcionamos a cada estado un significado en el contexto del problema, de modo muy parecido a lo que hicimos en el diseño de máquinas de estado temporizadas. Sin embargo, es

EXPLOSIÓN DE TÉRMINO DE PRODUCTO

En algunos casos, la necesidad de cubrir riesgos puede causar una explosión en el número de términos de producto en una realización de dos niveles de la lógica de excitación. Por ejemplo, supongamos que necesitamos un latch especializado con dos entradas de control, C1 y C2, y tres entradas de datos como antes. El latch estará "abierto" solamente si ambas entradas de control son 1, y podrá almacenar un 1 si cualquier entrada de datos es 1. La ecuación de excitación mínima es

$$Q^* = C1 \cdot C2 \cdot (D1 + D2 + D3) + (C1 \cdot C2)' \cdot Q$$
$$= C1 \cdot C2 \cdot D1 + C1 \cdot C2 \cdot D2 + C1 \cdot C2 \cdot D3 + C1' \cdot Q + C2' \cdot Q$$

Sin embargo, toma seis términos de consenso eliminar los riesgos (véase el ejercicio 7.76).

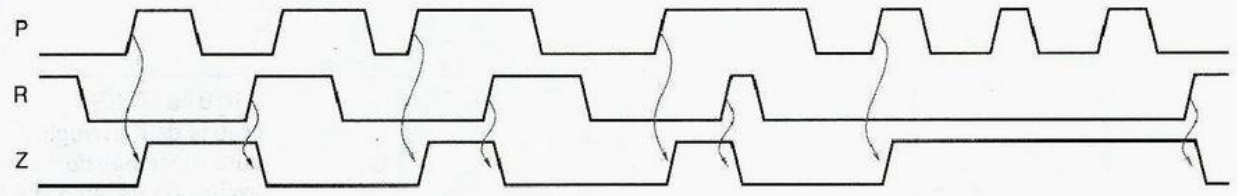


Figura 7-90 Comportamiento funcional típico de un circuito de captura de pulsos.

tabla de flujo primitiva

más fácil llegar a confundirse cuando se construye la tabla de flujo para un circuito secuencial de retroalimentación, porque no todo estado total es estable. Por consiguiente, el procedimiento recomendado es construir una *tabla de flujo primitiva*, una que solamente tiene un estado total estable en cada renglón. Puesto que hay solamente un estado estable por renglón, la salida puede mostrarse como una función únicamente del estado.

En una tabla de flujo primitiva, cada estado tiene un "significado" más preciso que el que tendría de otro modo, y la estructura de la tabla exhibe claramente la suposición de modo fundamental subyacente: las entradas cambian una a la vez, con tiempo suficiente entre los cambios para que el circuito se instale en un nuevo estado estable. Una tabla de flujo primitiva por lo regular tiene estados extra, pero más tarde podemos hacer los cálculos para minimizar el número de estados, una vez que tengamos una tabla de flujo que creamos se encuentre correcta.

Utilizaremos el problema siguiente, un circuito receptor o "de captura" de pulsos, para demostrar el diseño de tabla de flujo:

> Diseñe un circuito secuencial de retroalimentación con dos entradas, P(pulso) y R(restablecimiento) y una sola salida Z que sea normalmente 0. La salida debería establecerse a 1 cuando se presente una transición de 0 a 1 en P, y debería restablecerse a 0 siempre que R sea 1. El comportamiento funcional típico se muestra en la figura 7-90.

La figura 7-91 es una tabla de flujo primitiva para el receptor de pulsos. Repasemos el proceso de cómo fue desarrollada esta tabla.

Supongamos que el receptor de pulsos se encuentre inicialmente inactivo, siendo tanto P como R, 0; éste es el estado IDLE (inactivo), con Z = 0. En este estado, si ocurre un restablecimiento (R = 1), probablemente podríamos permanecer en el mismo estado, pero ya que se supone que esto es una tabla de flujo *primitiva*, creamos un nuevo estado, RES1, de modo que no tengamos dos estados totales estables en el mismo renglón. Por otra parte, si se presenta un pulso (P = 1) en el estado IDLE (inactivo), definitivamente vamos a querer ir a un estado diferente, que hemos nombrado PLS1, puesto que hemos capturado un pulso y debemos establecer la salida a 1. La combinación de entrada 11 no se considera en el estado IDLE, debido a la suposición del modo fundamental de que solamente una entrada cambia a la vez; suponemos que el circuito siempre lo hace para otro estado estable antes de que la combinación de entrada 11 pueda ocurrir.

A continuación, rellenamos las entradas del estado siguiente para el estado apenas creado RES1. Si el restablecimiento se retira, podemos regresar al estado IDLE. Si se presenta un pulso, debemos permanecer en un estado de "restablecimiento" ya que, de acuerdo con el diagrama de temporización, una transición de 0 a 1 que ocurre mientras R es 1 se ignora. De nueva cuenta, para mantener la tabla de flujo en forma primitiva, debemos crear un nuevo estado para este caso, RES2.

		P R				
Significado	S	00	01	11	10	Z
Inactivo (idle), esperando por pulso	IDLE	(IDLE)	RES1	—	PLS1	0
Restablecimiento (reset), ausencia de pulso	RES1	IDLE	(RES1)	RES2	—	0
Obtención de pulso, salida encendida	PLS1	PLS2	—	RES2	(PLS1)	1
Restablecimiento, obtención de pulso	RES2	—	RES1	(RES2)	PLSN	0
Salida de pulso, salida encendida	PLS2	(PLS2)	RES1	—	PLS1	1
Obtención de pulso, pero salida apagada	PLSN	IDLE	—	RES2	(PLSN)	0
				S*		

Figura 7-91 Tabla de flujo primitiva para circuito de captura de pulsos.

Ahora que tenemos un estado total estable en cada columna, podemos ser capaces de ir a los estados existentes para más transiciones, en lugar de definir siempre nuevos estados. Efectivamente, comenzando en el estado total estable PLS1/10, para R = 1 podemos ir a RES2, lo cual satisface el requerimiento de producir una salida 0. Por otro lado, ¿dónde deberíamos ir para P = 0? IDLE es un estado total estable en la columna 00, pero produce el valor equivocado de salida. En PLS1, tenemos un pulso y no hemos visto un restablecimiento, de manera que si el pulso se retira, deberíamos ir a un estado que todavía tenga Z = 1. De este modo, debemos crear un nuevo estado PLS2 para este caso.

En RES2, podemos ir con seguridad a RES1 si el pulso se retira. Sin embargo, tenemos que ser cuidadosos si el restablecimiento se retira, como se muestra en el diagrama de temporización. Puesto que ya hemos pasado la transición de 0 a 1 del pulso, no podemos ir al estado PLS1, ya que podría darnos una salida 1. En su lugar, creamos un nuevo estado PLSN con una salida 0.

Finalmente, podemos completar las entradas de estado siguiente para PLS2 y PLSN sin crear ningún estado nuevo. Note que, al iniciar en PLS2, oscilamos hacia atrás y hacia delante entre PLS2 y PLS1, y mantenemos una salida 1 continua si obtenemos una serie de pulsos sin que intervenga una entrada de restablecimiento.

*7.10.3 Minimización de la tabla de flujo

Como mencionamos con anterioridad, una tabla de flujo primitiva por lo regular tiene más estados que los requeridos. Sin embargo, existe un procedimiento formal, que se discute en las Referencias, para minimizar el número de estados en una tabla de flujo. Este procedimiento con frecuencia es complicado por la existencia de entradas sin importancia en la tabla de flujo.

Afortunadamente, nuestra tabla de flujo de ejemplo es suficientemente simple y pequeña para minimizarse por inspección. Los estados IDLE y RES1 producen la misma salida, y tienen la misma entrada de estado siguiente para combinaciones de entrada donde ambas son especificadas. Por lo tanto, son compatibles y se pueden reemplazar con un solo estado (IDLE) en una tabla de flujo reducida. Lo mismo puede

	P R				
S	00	01	11	10	Z
IDLE	(IDLE)	(IDLE)	RES	PLS	0
PLS	(PLS)	IDLE	RES	(PLS)	1
RES	IDLE	IDLE	(RES)	(RES)	0
			S*		

Figura 7-92
Tabla de flujo reducida para circuito de captura de pulsos.

decirse para los estados PLS1 y PLS2 (reemplazados por PLS) y para RES2 y PLSN (reemplazados por RES). La tabla de flujo reducida que resulta y que tiene solamente tres estados, se muestra en la figura 7-92.

*7.10.4 Asignación de estado libre de carreras

El siguiente paso algo creativo (léase "difícil") en el diseño de circuitos secuenciales de retroalimentación es encontrar una asignación libre de carreras de estados codificados para los estados nombrados en la tabla de flujo reducida. Recuerde de la sección 7.9.3 que se presenta una carrera cuando múltiples variables internas cambian de estado como resultado de un solo cambio de entrada. Un circuito secuencial basado en retroalimentación no debe contener ninguna carrera crítica; de otro modo, el circuito puede funcionar de manera impredecible. Como veremos, la eliminación de carreras a menudo hace necesario *incrementar* el número de estados en el circuito.

diagrama de adyacencia de estado

El potencial de un circuito para tener carreras en su tabla de transición puede ser analizado por medio de un *diagrama de adyacencia de estado* para su tabla de flujo. El diagrama de adyacencia de estado es un diagrama de estado simplificado que omite los lazos propios ("autolazos") y no muestra la dirección de otras transiciones (A → B se dibuja igual que B → A) o las combinaciones de entrada que las causan. La figura 7-93 es un ejemplo de tabla de flujo en modo fundamental y la figura 7-94(a) es el correspondiente diagrama de adyacencia de estado.

estados adyacentes

Se dice que dos estados son *adyacentes* si hay un arco entre ellos en el diagrama de adyacencia de estado. Para transiciones libre de carreras, los estados codificados adyacentes deben diferir en solamente un bit. Si dos estados A y B son adyacentes, no importa si la

	X Y			
S	00	01	11	10
A	(A)	B	(A)	B
B	(B)	(B)	D	(B)
C	(C)	A	A	(C)
D	(D)	B	(D)	C
			S*	

Figura 7-93
Ejemplo de tabla de flujo para el problema de asignación de estado.

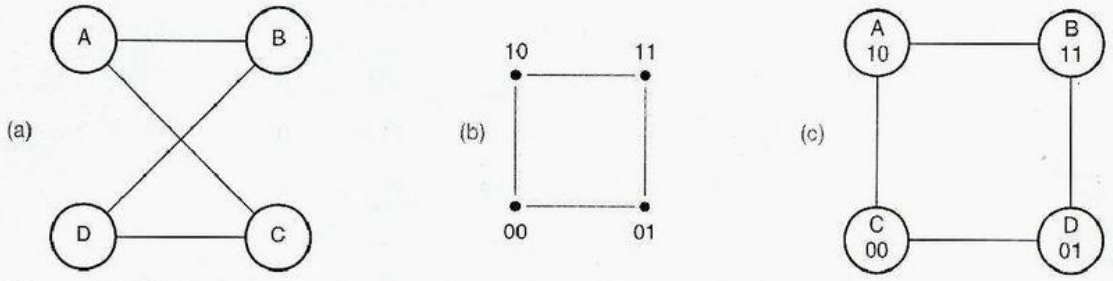


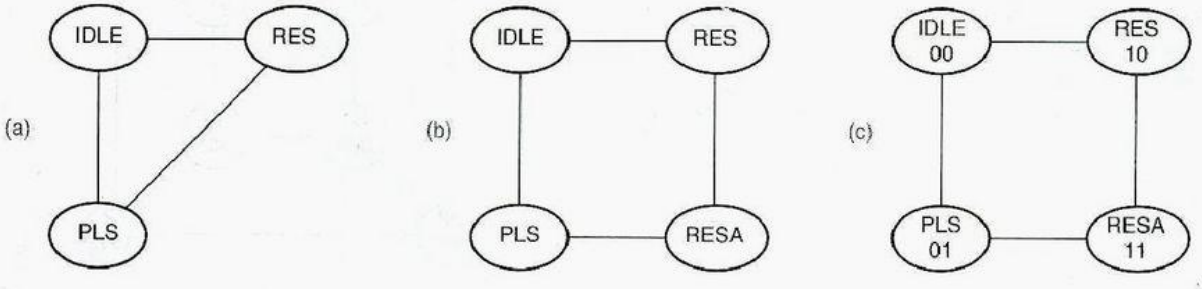
Figura 7-94 Ejemplo de asignación de estado: (a) diagrama de adyacencia, (b) un cubo 2; (c) uno de ocho posibles asignaciones de estado libre de carreras.

tabla de flujo original tenía transiciones desde A hacia B, de B hacia A, o de ambos tipos. Cualquiera de estas transiciones es una carrera si A y B difieren en más de una variable. Por eso no necesitamos mostrar la dirección de las transiciones en un diagrama de adyacencia de estado.

El problema de encontrar una asignación de estados libres de carreras para n variables de estado es equivalente al problema de mapear los nodos y arcos del diagrama de adyacencia sobre los nodos y arcos de un cubo n. En la figura 7-94, el problema es mapear el diagrama de adyacencia (a) sobre un cubo 2 (b). Se pueden identificar visualmente ocho maneras de hacer esto (cuatro rotaciones por dos flips), una de las cuales produce la asignación de estado mostrada en (c).

La figura 7-95(a) es un diagrama de adyacencia para nuestro circuito de "captura de pulsos", basado en la tabla de flujo reducida en la figura 7-92. Evidentemente, no hay manera de mapear este "triángulo" de estados sobre un cubo 2. En este punto, solamente podemos retroceder y modificar la tabla de flujo original. En particular, la tabla de flujo nos dice el estado destino que *finalmente* debemos alcanzar para cada transición, pero no nos evita ir a través de otros estados en el recorrido. Como se muestra en la figura 7-96, podemos crear un nuevo estado RESA y hacer la transición desde PLS a RES pasando a través de RESA. La tabla de estado modificada tiene el nuevo diagrama de adyacencia mostrado en la figura 7-95(b), el cual tiene muchas asignaciones de carrera libre posibles. Una tabla de transición basada en la asignación en (c) se muestra en la figura 7-98. Advierta que la transición PLS → RESA → RES será más lenta que las otras transiciones en

Figura 7-95 Diagramas de adyacencia para el receptor de pulsos; (a) diagrama de adyacencia; (b) un cubo 2; (c) número de ocho posibles asignaciones de estado libre de carreras.



	P R				
S	00	01	11	10	Z
IDLE	(IDLE)	(IDLE)	RES	PLS	0
PLS	(PLS)	IDLE	RESA	(PLS)	1
RESA	—	—	RES	—	–
RES	IDLE	IDLE	(RES)	(RES)	0
	S*				

Figura 7-96
Tabla de estado permitiendo una asignación libre de carrera para el circuito de captura de pulsos.

la tabla de flujo original ya que requiere de dos cambios de estado interno, con dos retardos de propagación a través de los lazos de retroalimentación.

Aun cuando agregamos un estado en el ejemplo anterior, todavía nos las arreglamos con sólo dos variables de estado. Sin embargo, en ocasiones podemos tener que agregar una o más variables de estado para hacer una asignación libre de carrera. La figura 7-97(a) muestra el peor diagrama de adyacencia posible para cuatro estados: cada estado es adyacente con respecto a cada uno de los otros estados. Evidentemente, este diagrama de adyacencia no se puede mapear sobre un cubo 2. Sin embargo, existe una asignación libre de carrera de estados para un cubo 3, mostrado en (b), donde cada estado en la tabla de flujo original es representado por dos estados equivalentes en la tabla de estado final. Ambos estados en un par, tal como A1 y A2, son equivalentes y producen la misma salida. Cada estado es adyacente a uno de los estados en cada uno de los otros pares, de modo que una transición libre de carrera puede seleccionarse para cada entrada de estado siguiente.

MANEJO DEL CASO DE ASIGNACIÓN GENERAL

En el caso general de una tabla de flujo con 2^n renglones, puede demostrarse que una asignación libre de carrera se obtiene utilizando $2n - 1$ variables de estado (vea las Referencias). Sin embargo, no hay muchas aplicaciones para circuitos en modo fundamental con más de unos cuantos estados, de modo que el caso general es de apenas interés académico.

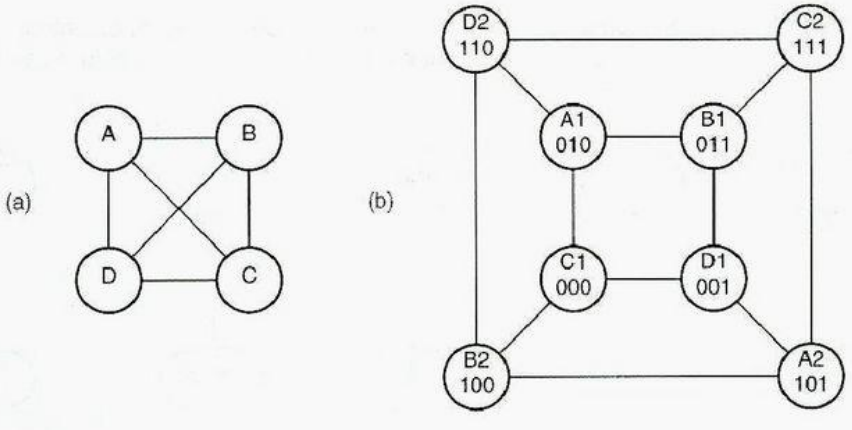


Figura 7-97
Un escenario del peor de los casos: (a) diagrama de adyacencia de 4 estados; (b) asignación que utiliza pares de estados equivalentes.

	P R				
Y1 Y2	00	01	11	10	Z
00	(00)	(00)	10	01	0
01	(01)	00	11	(01)	1
11	—	—	10	—	–
10	00	00	(10)	(10)	0
	Y1* Y2*				

Figura 7-98
Tabla de transición libre de carrera para el circuito de captura de pulsos.

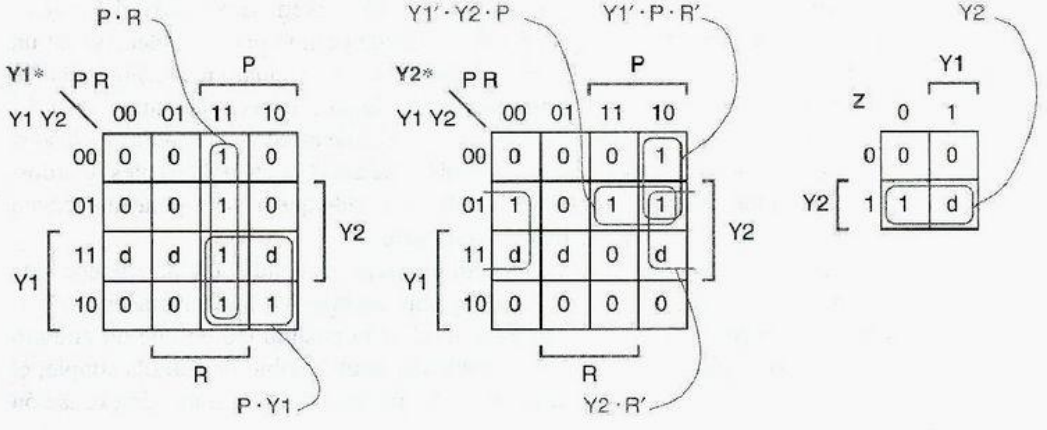
*7.10.5 Ecuaciones de excitación

Una vez que tenemos una tabla de transición libre de carrera para un circuito, sólo podemos "poner manos a la obra" para obtener las ecuaciones de excitación para los lazos de retroalimentación. La figura 7-99 muestra mapas de Karnaugh que se obtienen de la figura 7-98, la tabla de transición del receptor de pulsos. Note que las entradas de estado siguiente "sin importancia" y salida dan origen a las correspondientes entradas en los mapas, simplificando la lógica de excitación y salida. Las ecuaciones mínimas resultantes de salida y excitación de suma de productos son como se ve a continuación:

$$Y1* = P \cdot R + P \cdot Y1$$
$$Y2* = Y2 \cdot R' + Y1' \cdot Y2 \cdot P + Y1' \cdot P \cdot R'$$
$$Z = Y2$$

Recuerde que la lógica de excitación en un circuito secuencial de retroalimentación debe estar libre de riesgos. Las expresiones de suma de productos que obtuvimos no tienen riesgos, y además son mínimas. El diagrama lógico de la figura 7-100 utiliza estas expresiones para construir el circuito de captura de pulsos.

Figura 7-99 Mapas de Karnaugh para excitación del receptor de pulsos y lógica de salida.



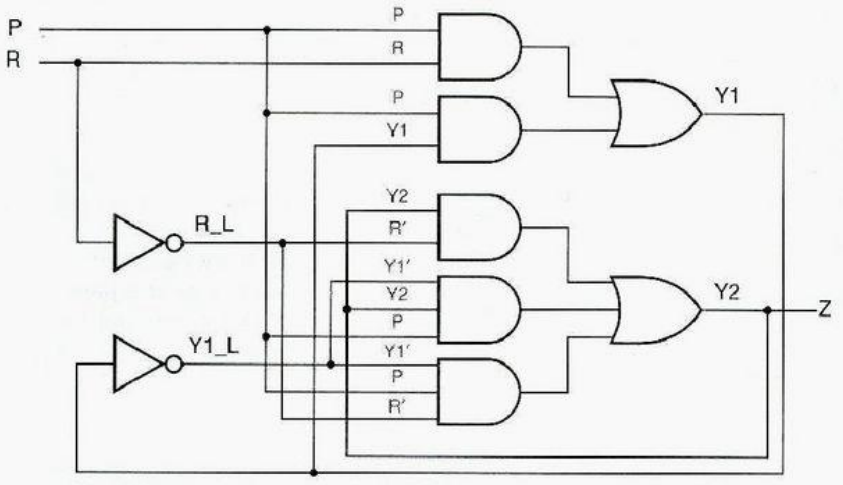


Figura 7-100
Circuito de captura de pulsos.

*7.10.6 Riesgos esenciales

Después de todo este esfuerzo, usted pensaría que tendríamos un circuito de captura de pulsos que funcionaría confiablemente todo el tiempo. Desgraciadamente, todavía no basta. Un circuito en modo fundamental generalmente satisface cinco requerimientos para un funcionamiento adecuado:

1. Solamente una señal de entrada puede cambiar a la vez, con una frontera mínima entre sucesivos cambios de entrada.
2. Debe existir un retardo de propagación máximo a través de la lógica de excitación y las trayectorias de retroalimentación; este máximo debe ser menor que el tiempo entre cambios de entrada sucesivos.
3. La asignación de estado (tabla de transición) debe estar libre de carreras críticas.
4. La lógica de excitación debe estar libre de riesgos.
5. El retardo de propagación mínimo a través de la lógica de excitación y trayectorias de retroalimentación debe ser mayor que el sesgo de tiempo máximo a través de la "lógica de entrada".

Sin el primer requerimiento, sería imposible satisfacer la premisa principal del funcionamiento en modo fundamental: que el circuito tenga tiempo para establecerse en un estado total estable entre cambios de entrada sucesivos. El segundo requerimiento dice que la lógica de excitación es suficientemente rápida para hacerlo justamente así. El tercer requerimiento asegura que se hagan los cambios de estado apropiados, incluso si los circuitos de excitación para diferentes variables de estado tienen diferentes retardos. El cuarto requerimiento asegura que las variables de estado que no se supone que tengan cambios en una transición en particular, no lo hagan.

riesgo esencial

El último requerimiento trata con sutiles errores dependientes del tiempo que pueden ocurrir en circuitos en modo fundamental, incluso los que satisfacen los primeros cuatro requerimientos. Un *riesgo esencial* es la posibilidad de que un circuito vaya a un estado siguiente erróneo como resultado de un cambio de entrada simple; el error se presenta si el cambio de entrada no se ve por todos los circuitos de excitación

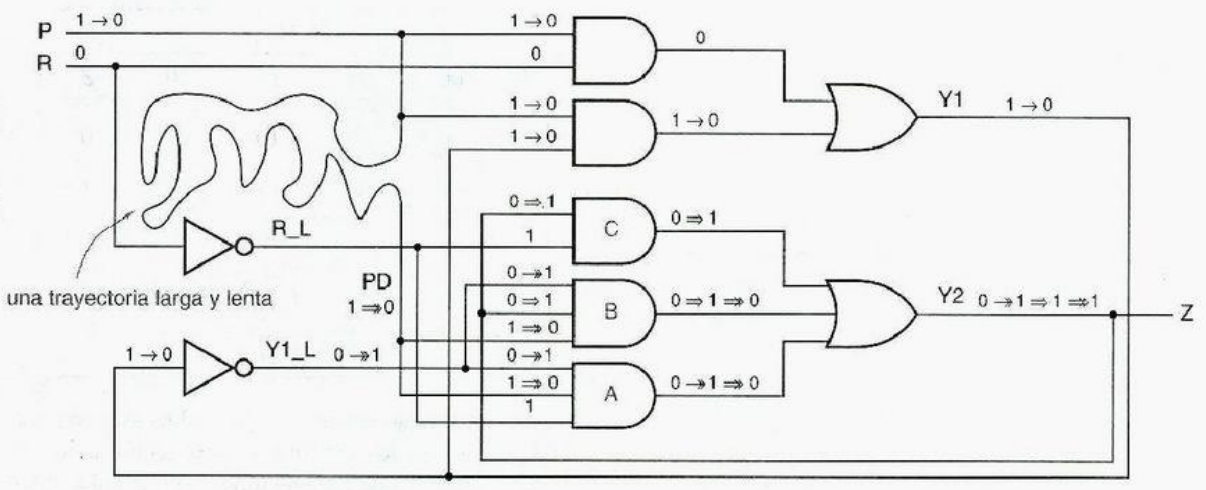


Figura 7-101 Condiciones físicas en el circuito de captura de pulsos para exhibir un riesgo esencial.

antes que la transición (o transiciones) de variables de estado resultante(s) se propaguen de regreso a las entradas de los circuitos de excitación. En un mundo donde "lo más rápido es lo mejor" es la regla habitual, un diseñador en ocasiones puede tener que *hacer más lenta* la lógica de excitación para disimular estos riesgos.

Los riesgos esenciales se explican mejor en términos de un ejemplo, nuestro circuito de captura de pulsos. Supongamos que construimos nuestro circuito en un PCB o un chip, y que nosotros (o más probablemente, nuestro sistema CAD) involuntariamente conectamos la señal de entrada P a través de una trayectoria larga y lenta en el punto mostrado en la figura 7-101. Supongamos que este retardo es mayor que el retardo de propagación de una lógica de excitación AND-OR.

Ahora considere lo que puede pasar si P R = 10, el circuito está en estado interno 10 y P cambia de 1 a 0. De acuerdo con la tabla de transición, repetida en la figura 7-102, el circuito debería ir al estado interno 00, y eso es. Pero examinemos el funcionamiento real del circuito, como se describe en la figura 7-101:

- (Cambios mostrados con "→") La primera cosa que ocurre después de que P cambia, es que Y1 cambia de 1 a 0. Ahora el circuito se encuentra en estado interno 00.
- (Cambios mostrados con "→→") Y1_L cambia de 0 a 1. El cambio en Y1_L en la compuerta AND A causa que su salida vaya a 1, lo cual a su vez obliga a Y2 para ir a 1. Caramba, ahora el circuito se encuentra en estado interno 01.
- (Cambios mostrados con "⇒") El cambio en Y2 en las compuertas AND B y C provoca que sus salidas vayan a 1, reforzando la salida 1 en Y2. Todo este tiempo, hemos estado esperando que el cambio de 1 a 0 en P aparezca en el punto PD.
- (Cambios mostrados con "⇒⇒") Finalmente, PD cambia de 1 a 0, forzando las salidas de las compuertas AND A y B a 0. Sin embargo, la compuerta AND C todavía tiene una salida 1, y el circuito permanece en el estado 01: *el estado equivocado.*

	P R				
Y1 Y2	00	01	11	10	Z
00	(00)	(00)	10	01	0
01	(01)	00	11	(01)	1
11	—	—	10	—	—
10	00	00	(10)	(10)	0
	Y1* Y2*				

Figura 7-102
Tabla de transición para el circuito de captura de pulsos, exhibiendo un riesgo esencial.

sesgo de temporización

La única manera de evitar este comportamiento erróneo en general es asegurar que los cambios en P lleguen a las entradas de todos los circuitos de excitación antes de que cualquier cambio en las variables de estado lo haga. De este modo, la inevitable diferencia en los tiempos de llegada de entrada, denominado *sesgo de temporización*, debe ser menor que el retardo de propagación de los circuitos de excitación y lazos de retroalimentación. Este requerimiento de temporización generalmente puede satisfacerse solamente mediante un diseño cuidadoso *al nivel del circuito eléctrico*.

En el circuito de ejemplo, parecería que el riesgo es fácilmente disimulado, incluso por ingenieros no eléctricos, puesto que el diseñador sólo necesita asegurar que un alambre recto tiene un retardo de propagación más corto que una estructura AND-OR: fácil en la mayoría de las tecnologías.

Con todo, muchos circuitos secuenciales de retroalimentación, tales como el flip-flop D disparado por flanco TTL en la figura 7-19, tienen riesgos esenciales en los que las trayectorias de sesgo de entrada incluyen inversores. En tales casos, debe garantizarse que los inversores de entrada sean más rápidos que la lógica de excitación; eso no es tan trivial ni en el diseño al nivel de tarjeta ni de CI. Por ejemplo, si el circuito de excitación en la figura 7-101 se construye con compuertas AND-OR-INVERSOR, el retardo de los cambios de entrada hasta Y1_L podría ser en realidad muy corto, tan corto como el retardo a través de un único inversor.

Los riesgos esenciales pueden encontrarse en la mayoría pero no en todos los circuitos en modo fundamental. Existe una regla fácil para detectarlos: de hecho, ésta es la definición de "riesgo esencial" en algunos textos:

- Una tabla de flujo en modo fundamental contiene un riesgo esencial para un estado total estable S y una variable de entrada X si, comenzando en el estado S, el estado total estable alcanzado después de tres transiciones sucesivas en X es diferente del estado total estable alcanzado después de una transición en X.

¡ESTOS RIESGOS SON, BUENO, ESENCIALES!

Los riesgos esenciales son llamados "esenciales" porque son inherentes en la tabla de flujo para una función secuencial particular y aparecerá en cualquier realización de circuito de esa función. Solamente se pueden ocultar al controlar los retardos en el circuito. Compare con los riesgos estáticos en la lógica combinacional, donde podíamos eliminar riesgos al agregar términos de consenso a una expresión lógica.

De este modo, el riesgo esencial en el receptor de pulsos es detectado por las flechas en la figura 7-102, comenzando en el estado interno 10 con P R = 10.

Un circuito en modo fundamental debe tener al menos tres estados para tener un riesgo esencial, así que los latchs no los tienen. Por otra parte, todos los flip-flops, que son circuitos que realizan el muestreo de sus entradas en un flanco de reloj, sí los tienen.

*7.10.7 Resumen

En resumen, se utilizan los pasos siguientes para diseñar un circuito secuencial de retroalimentación:

1. Construya una tabla de flujo primitiva a partir de la descripción en palabras del circuito.
2. Minimice el número de estados en la tabla de flujo.
3. Encuentre una asignación de estados codificados libre de carreras para estados nombrados, agregando estados auxiliares o dividiendo estados como se vaya requiriendo.
4. Construya la tabla de transición.
5. Construya mapas de excitación y encuentre una realización libre de riesgos de las ecuaciones de excitación.
6. Verifique la existencia de riesgos esenciales. Modifique el circuito si es necesario para asegurar que los retardos de excitación y retroalimentación mínimos sean mayores que los retardos máximos de inversor u otra lógica de entrada.
7. Dibuje el diagrama lógico.

También advierta que algunos circuitos rutinariamente violan la suposición básica del modo fundamental que cambia las entradas una a un tiempo. Por ejemplo, en un flip-flop D disparado por flanco positivo, la entrada D puede cambiar al mismo tiempo que CLK cambia de 1 a 0, y el flip-flop todavía funciona apropiadamente. Ciertamente esto mismo no puede decirse en la transición de 0 a 1 de CLK. Tales situaciones requieren del análisis del circuito y tabla de transición caso por caso si se quiere garantizar un funcionamiento adecuado en "casos especiales".

UNA PREGUNTA FINAL Dada la dificultad del diseño de circuitos en modo fundamental que funcionan apropiadamente, dejando sólo los que son rápidos o compactos, ¿cómo llegaría alguien a proponer el flip-flop D comercial de 6 compuertas y 8 estados en la figura 7-20? ¡No me pregunten, yo no lo sé!

7.11 Características de diseño de circuito secuencial ABEL

7.11.1 Salidas registradas

ABEL tiene varias características que apoyan el diseño de circuitos secuenciales. Como demostraremos en la sección 8.3, el usuario puede configurar la mayor parte de las salidas PLD, para que sean *salidas registradas* que proporcionan un flip-flop D siguiendo la lógica AND-OR, como en la figura 7-103. Para configurar una o más salidas para ser

salida registrada

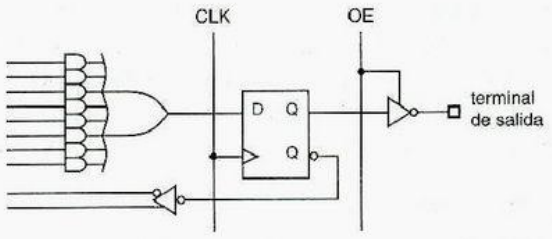


Figura 7-103
Salida registrada PLD.

reg

registradas, las declaraciones de terminal de un programa ABEL normalmente deben contener una cláusula `istype` que utiliza la palabra clave "`reg`" (en vez de "`com`") para cada salida registrada. La tabla 7-22 es un programa de ejemplo que tiene tres salidas registradas y dos salidas combinacionales.

Como se sugiere en la figura 7-103, una salida registrada tiene por lo menos otros dos atributos asociados con ella. El separador de tres estados, que controla la terminal de salida, tiene una entrada OE de habilitación de salida y el mismo flip-flop tiene una entrada de reloj CLK. Como se muestra en la tabla 7-22, las señales que controlan estas entradas son especificadas en la sección de ecuaciones del programa. Cada señal de entrada se especifica como el correspondiente nombre de la señal de salida principal, seguido por un sufijo de atributo, *.CLK* o *.OE*. Algunos PLD tienen flip-flops con entradas controlables adicionales. Por ejemplo, las entradas de restablecimiento y prestablecimiento asincrónicos tienen sufijos de atributo `.AR` y `.AP`; las entradas de restablecimiento y prestablecimiento sincrónicas utilizan sufijos `.SR` y `.SP`. Algunos PLD proporcionan otros tipos de flip-flops aparte del D; sus entradas están especificadas con sufijos como `.J` y `.K`.

.CLK
.OE
.AR
.AP
.SR
.SP

Dentro de la sección `equations` del programa ABEL, los valores lógicos para salidas registradas son establecidos empleando el *operador de asignación temporizado, :=*. Cuando el PLD es compilado, la expresión en el lado derecho será aplicada a la entrada D del flip-flop de salida. Todas las mismas reglas para las salidas combinacionales se aplican al control de polaridad de salida, generación del conjunto de iniciación, elementos sin importancia y así sucesivamente. En la tabla 7-22, los bits de estado Q1-Q3 son salidas registradas, de manera que utilizan la asignación temporizada, "`:=`". Una máquina con salidas canalizadas (figura 7-37), utilizaría en su lugar asignación temporizada para tales salidas.

operador de asignación temporizado, :=

La sintaxis de la tabla de verdad de ABEL (tabla 4-16) también puede ser usada con salidas registradas. La única diferencia es que el operador "`->`" entre los elementos de entrada y salida se cambia a "`:>`".

operador de tabla de verdad temporizada, :>

¿ES ESENCIAL `istype`? Los dispositivos más antiguos, tales como la familia PAL16Rx, contienen una mezcla de salidas registradas y combinacionales preasignadas y fijas y no son configurables. Con estos dispositivos, el compilador puede deducir cada tipo de salida de su número de terminal y la instrucción `istype` no es necesaria. Incluso con salidas configurables, algunos compiladores pueden deducir correctamente el tipo de salida de las ecuaciones. Con todo, es una buena idea incluir la información `istype`, tanto como una verificación doble como para mejorar la transportabilidad del diseño.

Tabla 7-22 Programa ABEL que utiliza salidas registradas.

```
module CombLock
Title 'Combination-Lock State Machine'

" Input and Outputs
X, CLOCK        pin;
UNLK, HINT      pin istype 'com';
Q1, Q2, Q3      pin istype 'reg';

Q = [Q1..Q3];

Equations

Q.CLK = CLOCK; Q.OE = 1;

" State variables
Q1 := Q1 & !Q2 & X  #  !Q1 & Q2 & Q3 & !X  #  Q1 & Q2 & !Q3;
Q2 := !Q2 & Q3 & X  #  Q2 & !Q3 & X;
Q3 := Q1 & !Q2 & !Q3  #  Q1 & Q3 & !X  #  !Q2 & !X
      #  !Q1 & !Q3 & !X  #  Q2 & !Q3 & X;

" Mealy outputs
UNLK = Q1 & Q2 & Q3 & !X;
HINT = !Q1 & !Q2 & !Q3 & !X  #  Q1 & !Q2 & X  #  !Q2 & Q3 & X
       #  Q2 & Q3 & !X  #  Q2 & !Q3 & X;

end CombLock
```

También se pueden diseñar circuitos secuenciales de retroalimentación en ABEL sin utilizar ninguna de las características de circuito secuencial del lenguaje. Por ejemplo, en la sección 8.2.6 mostraremos cómo especificar cerrojos empleando ABEL.

7.11.2 Diagramas de estado

El ejemplo de máquina de estado en la subsección anterior es sólo una transcripción de la máquina de combinación asegurada que sintetizamos a mano en la sección 7.4.6. Sin embargo, la mayoría de los lenguajes de programación de PLD tienen una notación para definir, documentar y sintetizar máquinas de estado directamente, sin escribir siempre una tabla de estado, transición o excitación u obtener ecuaciones de excitación a mano. Una notación de esta clase es llamada *lenguaje de descripción de máquina de estado*. En ABEL, esta notación es conocida como un "diagrama de estado" y el compilador de ABEL hace todo el trabajo de generar las ecuaciones de excitación que realiza la máquina especificada.

lenguaje de descripción de máquina de estado

En ABEL, la palabra clave `state_diagram` indica el principio de una definición de máquina de estado. La tabla 7-23 muestra la estructura textual de un "diagrama de estado" ABEL. Aquí *vector-estado* es un conjunto de ABEL que enumera las variables de estado de la máquina. Si hay n variables en el conjunto, entonces la máquina tiene 2^n estados posibles correspondientes a las 2^n asignaciones diferentes de valores constantes para variables en el conjunto. Los estados son habitualmente nombres simbólicos dados en un programa ABEL; esto hace fácil intentar asignaciones diferentes simplemente cambiando las definiciones constantes.

`state_diagram`

vector-estado

Tabla 7-23
Estructura de un "diagrama de estado" en ABEL.

```
state_diagram vector-estado
state valor-estado 1 :instrucción de transición;
state valor-estado 2 : instrucción de transición;
...
state valor-estado 2^n : instrucción de transición;
```

El conjunto de iniciación para cada variable de estado es creado de acuerdo a la información en el "diagrama de estado". Si una variable de estado también aparece en el lado izquierdo de una ecuación en la sección `equations`, los efectos son acumulativos (vea el cuadro en la página 739 para una aplicación útil de este comportamiento). La palabra clave `state` indica que los estados siguientes y salidas actuales para un estado actual en particular están por ser definidos; un *valor-estado* es una constante que define valores estado-variable para el estado actual. Una *instrucción de transición* define los posibles estados siguientes para el estado actual.

state
valor-estado
instrucción de transición

ABEL tiene dos instrucciones de transición que se utilizan con bastante frecuencia. La *instrucción GOTO* especifica incondicionalmente el estado siguiente, por ejemplo, "GOTO INIT". La *instrucción IF* define los posibles estados siguientes como una función de unas expresiones lógicas arbitrarias. (Hay también una instrucción `CASE` rara vez empleada que no cubriremos.)

instrucción GOTO
instrucción IF

La tabla 7-24 muestra la sintaxis de la instrucción `IF` de ABEL. Aquí *EstadoVerdadero* (true state) y *EstadoFalso* (false state) son valores de estado a los que la máquina llegará si *ExpresiónLógica* es verdadera o falsa, respectivamente. Estas instrucciones pueden estar anidadas; *EstadoFalso* (false state) puede ser por sí mismo otra instrucción `IF`, y *EstadoVerdadero* (true state) puede ser una instrucción `IF` si se encuentra encerrada entre corchetes. Cuando son posibles múltiples estados siguientes, una estructura `IF-THEN-ELSE` elimina las ambigüedades que puedan presentarse en diagramas de estado trazados a mano, donde las condiciones de transición que dejan un estado pueden traslaparse (sección 7.5).

Nuestro primer ejemplo que utiliza la capacidad de "diagrama de estado" de ABEL está basado en nuestro primer ejemplo de diseño de máquina de estado de la sección 7.4.1 en la página 564. Una tabla de estado para esta máquina se desarrolló en la figura 7-49 de la página 567. Está adaptado para ABEL en la tabla 7-25. Deberían observarse varias características de este programa:

- La definición de `QSTATE` utiliza tres variables para codificar estado.
- Las definiciones de `INIT–XTRA3` determinan las codificaciones de estado individuales.
- Las instrucciones `IF-THEN-ELSE` se encuentran anidadas. Un estado siguiente particular puede aparecer en múltiples lugares en un conjunto de cláusulas `IF-THEN-ELSE` anidadas (por ejemplo, véanse los estados `OK0` y `OK1`).

Tabla 7-24
Estructura de una instrucción `IF` de ABEL.

```
IF ExpresiónLógica THEN
   EstadoVerdadero
ELSE
   EstadoFalso;
```

Tabla 7-25
Un ejemplo de notación de diagrama de estado de ABEL.

```
module SMEX1
title 'PLD Version of Example State Machine'

" Input and output pins
CLOCK, RESET_L, A, B            pin;
Q1..Q3                          pin istype 'reg';
Z                               pin istype 'com';

" Definitions
QSTATE = [Q1,Q2,Q3];           " State variables
INIT   = [ 0, 0, 0];
A0     = [ 0, 0, 1];
A1     = [ 0, 1, 0];
OK0    = [ 0, 1, 1];
OK1    = [ 1, 0, 0];
XTRA1  = [ 1, 0, 1];
XTRA2  = [ 1, 1, 0];
XTRA3  = [ 1, 1, 1];
RESET = !RESET_L;

state_diagram QSTATE

state INIT: IF RESET THEN INIT
            ELSE IF (A==0) THEN A0
            ELSE A1;

state A0:   IF RESET THEN INIT
            ELSE IF (A==0) THEN OK0
            ELSE A1;

state A1:   IF RESET THEN INIT
            ELSE IF (A==0) THEN A0
            ELSE OK1;

state OK0:  IF RESET THEN INIT
            ELSE IF (B==1)&(A==0) THEN OK0
            ELSE IF (B==1)&(A==1) THEN OK1
            ELSE IF (A==0) THEN OK0
            ELSE IF (A==1) THEN A1;

state OK1:  IF RESET THEN INIT
            ELSE IF (B==1)&(A==0) THEN OK0
            ELSE IF (B==1)&(A==1) THEN OK1
            ELSE IF (A==0) THEN A0
            ELSE IF (A==1) THEN OK1;

state XTRA1: GOTO INIT;
state XTRA2: GOTO INIT;
state XTRA3: GOTO INIT;

equations

QSTATE.CLK = CLOCK; QSTATE.OE = 1;
Z = (QSTATE == OK0) # (QSTATE == OK1);

END SMEX1
```

Tabla 7-26
Ecuaciones reducidas para el PLD SMEX1.

```
Q1 := (!Q2.FB & !Q3.FB & RESET_L
     # Q1.FB & RESET_L);
Q1.C = (CLOCK);
Q1.OE = (1);

Q2 := (Q1.FB & !Q3.FB & RESET_L & !A
     # Q1.FB & Q3.FB & RESET_L & A
     # Q1.FB & Q2.FB & RESET_L & B);
Q2.C = (CLOCK);
Q2.OE = (1);

Q3 := (!Q2.FB & !Q3.FB & RESET_L & A
     # Q1.FB & RESET_L & A);
Q3.C = (CLOCK);
Q3.OE = (1);

Z = (Q2 & Q1);
```

- Se utilizaron expresiones como "(B==1)*(A==0)" en lugar de equivalentes como "B*!A" sólo porque las primeras son un poco más legibles.
- La primera instrucción IF en cada uno de los estados INIT–OK1 asegura que la máquina vaya al estado INIT si se afirma RESET.
- Las ecuaciones de estado siguiente están asignadas para XTRA1–XTRA3 para asegurar que la máquina se dirige a un estado "seguro" si de alguna manera llega a un estado sin utilizar.
- La ecuación simple, en la sección "equations" del programa, determina el comportamiento de la salida tipo Moore.

La tabla 7-26 muestra las ecuaciones resultantes de excitación y de salida producidas por el compilador ABEL (las ecuaciones de polaridad inversa no se muestran). Note el uso de nombres de variable como "Q1.FB" en los lados derechos de las ecuaciones. Aquí, el sufijo de atributo ".FB" se refiere a la señal de "retroalimentación" en el arreglo AND-OR que viene de la salida Q del flip-flop. Esto se efectúa para hacer evidente que la señal está llegando del flip-flop, no de la terminal de salida del PLD correspondiente, la cual puede ser seleccionada en algunos PLD complejos. Como se muestra en la figura 7-104,

ÚSELO O NO

La estructura IF-THEN-ELSE de ABEL elimina la ambigüedad de la transición que puede presentarse en los diagramas de estado. Sin embargo, la cláusula ELSE de una instrucción IF es opcional. Si se omite, el estado siguiente para algunas combinaciones de entrada quedará sin especificar. Habitualmente, ésta no es la intención del diseñador.

No obstante, si usted puede garantizar que las combinaciones de entrada sin especificar nunca se presentarán, usted puede ser capaz de reducir el tamaño de la lógica de transición. Si la directiva @DCSET está dada, el compilador ABEL trata las salidas de transición para las combinaciones estado/entrada sin especificar como "sin importancia". Además, trata *todas* las transiciones fuera de los estados sin utilizar como "sin importancia".

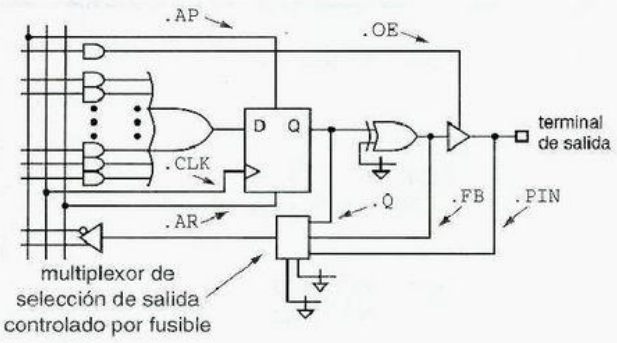


Figura 7-104
Capacidad de selección de salida en un PLD complejo.

ABEL en realidad le permite seleccionar entre tres posibles valores en el lado derecho de una ecuación que utiliza un sufijo de atributo en el nombre de la señal:

`.Q`	La terminal de salida del flip-flop real antes de cualquier inversión programable.	*.Q*
`.FB`	Un valor igual al valor que tendría laterminal de salida si se habilitara.	*.FB*
`.PIN`	La señal real en la terminal de salida del PLD. Esta señal está flotando o es controlada por otro dispositivo si el controlador de triple estado no está habilitado.	*.PIN*

Obviamente, el valor `.PIN` no debe utilizarse en una ecuación de excitación de máquina de estado, puesto que no siempre se garantiza que sea igual a la variable de estado.

A pesar del uso de "lenguaje de alto nivel", el autor del programa aún tendría que referirse a la tabla original construida a mano en la figura 7-49 para presentar la versión ABEL en la tabla 7-25. Un enfoque diferente se muestra en la tabla 7-27. Este programa fue desarrollado directamente a partir de la descripción en palabras de la máquina de estado, que repetimos a continuación:

> Diseñe una máquina de estado sincrónica temporizada con dos entradas, A y B, y una salida simple Z, que es 1 si:
>
> – A tiene el mismo valor en cada uno de los dos tics de reloj anteriores, *o*
> – B ha sido 1 desde la última vez que la primera condición fue verdadera.
>
> De otra manera, la salida debería ser 0.

Una idea clave en el nuevo enfoque es eliminar el último valor de `A` de las definiciones de estado, y en lugar de tener un flip-flop por separado que se mantenga al tanto del mismo (`LASTA`). Entonces solamente dos estados no `INIT` deben definirse: `LOOKING` ("seguir en busca de una coincidencia") y `OK` ("obtener una coincidencia o que `B` ya sido 1 desde la última coincidencia"). La salida `Z` es una decodificación combinacional simple del estado `OK`.

EL OPERANDO FANTASMA ("DE LA OPERA")

Por lo regular, los CPLD reales tienen solamente un multiplexor de selección de salida de dos entradas, y omiten la entrada `.FB` mostrada en la figura 7-104. Cuando una ecuación pide una señal con el atributo `.FB`, el compilador ABEL utiliza la señal `.Q` correspondiente y simplemente la ajusta con la inversión apropiada (o no).

Tabla 7-27
Un programa ABEL más "natural" para la máquina de estado de ejemplo.

```
module SMEX2
title 'Alternate Version of Example State Machine'

" Input and output pins
CLOCK, RESET_L, A, B            pin;
LASTA, Q1, Q2                   pin istype 'reg';
Z                               pin istype 'com';

" Definitions
QSTATE  = [Q1,Q2];               " State variables
INIT    = [ 0, 0];               " State encodings
LOOKING = [ 0, 1];
OK      = [ 1, 0];
XTRA    = [ 1, 1];
RESET = !RESET_L;

state_diagram QSTATE

state INIT:     IF RESET THEN INIT ELSE LOOKING;

state LOOKING: IF RESET THEN INIT
               ELSE IF (A == LASTA) THEN OK
               ELSE LOOKING;

state OK:       IF RESET THEN INIT
               ELSE IF B THEN OK
               ELSE IF (A == LASTA) THEN OK
               ELSE LOOKING;

state XTRA:     GOTO INIT;

equations
LASTA.CLK = CLOCK; QSTATE.CLK = CLOCK; QSTATE.OE = 1;

LASTA := A;
Z = (QSTATE == OK);

END SMEX2
```

*7.11.3 Memoria de estado externa

En algunas situaciones la memoria de estado de una máquina de estado basada en PLD puede mantenerse en flip-flops externos al PLD. ABEL proporciona una versión especial de la instrucción `state_diagram` para manejar esta situación:

`state_diagram` *variables-estado-actual* `->` *variables-estado-siguiente*

variables-estado-actual
variables-estado-siguiente

Aquí, *variables-estado-actual* es un conjunto ABEL que enumera las señales de entrada que representan el estado actual de la máquina, y *variables-estado-siguiente* es un conjunto que enumera las señales de salida correspondientes que son la excitación para los flip-flops D externos que mantienen el estado de la máquina, por ejemplo,

```
state_diagram [CURQ1, CURQ2] -> [NEXTQ1, NEXTQ2]
```

```
state_diagram variables-estado
state valor-estado 1 :
        ecuación opcional;
        ecuación opcional;
        ...
        instrucción de transición;
state valor-estado 2 :
        ecuación opcional;
        ecuación opcional;
        ...
        instrucción de transición;
...
state valor-estado 2^n :
        ecuación opcional;
        ecuación opcional;
        ...
        instrucción de transición;
```

Tabla 7-28
Estructura de un diagrama de estado ABEL con salidas de Moore definidas.

*7.11.4 Especificación de salidas de Moore

La salida Z en nuestra máquina de estado (del ejemplo) es una salida de Moore, solamente una función de estado, y definimos esta salida en las tablas 7-25 y 7-27 aplicando una ecuación apropiada en la sección equations del programa. De manera alternativa, ABEL permite que las salidas de Moore sean especificadas junto con las mismas definiciones de estado. La instrucción de transición en una definición state puede estar precedida por una o más ecuaciones opcionales, como se ilustra en la tabla 7-28. Para hacer uso de esta capacidad con la máquina en la tabla 7-27, por ejemplo, eliminaríamos la ecuación Z en la sección equations y volveríamos a escribir el diagrama de estado como se muestra en la tabla 7-29.

Como en otras ecuaciones ABEL, cuando una variable tal como Z aparece en el lado izquierdo de múltiples ecuaciones, los lados derechos se operan juntos con los OR para formar el resultado final (como se discutió en la sección 4.6.3). También adviértase

```
state_diagram QSTATE

state INIT:     Z = 0;
                IF RESET THEN INIT ELSE LOOKING;

state LOOKING:  Z = 0;
                IF RESET THEN INIT
                ELSE IF (A == LASTA) THEN OK
                ELSE LOOKING;

state OK:       Z = 1;
                IF RESET THEN INIT
                ELSE IF B THEN OK
                ELSE IF (A == LASTA) THEN OK
                ELSE LOOKING;

state XTRA:     Z = 0;
                GOTO INIT;
```

Tabla 7-29
Máquina de estado con definiciones de salida Moore integradas.

que Z todavía está especificada como una salida combinacional, no registrada. Si Z fuera una salida registrada, el valor de salida deseado se presentaría un tic de reloj después que la máquina visitara el estado correspondiente.

*7.11.5 Especificación de salidas Mealy y canalizadas con WITH

instrucción WITH

Algunas salidas de máquina de estado son funciones de las entradas así como también del estado. En la sección 7.3.2 las llamamos salidas de Mealy o salidas canalizadas, dependiendo de si se presentaban inmediatamente en un cambio de entrada o solamente después de un flanco de reloj. La *instrucción WITH* de ABEL proporciona una manera de especificar estas salidas juntas con los estados siguientes, en vez de por separado en la sección `equations` del programa.

Como se muestra en la tabla 7-30, la sintaxis de la instrucción `WITH` es muy sencilla. Cualquier valor de estado siguiente que sea parte de una instrucción de asignación puede estar seguido por la palabra clave `WITH` y una lista de ecuaciones entre corchetes que son "ejecutadas" por la transición especificada. Formalmente, sea "E" una expresión de excitación que sea verdadera solamente cuando la transición especificada vaya a tomarse. Entonces para cada ecuación en la lista entre corchetes de `WITH`, el lado derecho se opera con ANDs con E y se asigna al lado izquierdo. Las ecuaciones pueden emplear ya sea asignación sin temporización o con temporización para crear salidas tipo Mealy o canalizadas, respectivamente.

Tabla 7-30
Estructura de la instrucción `WITH` de ABEL.

```
estado-siguiente WITH {
        ecuación;
        ecuación;
        ...
}
```

Desarrollamos una máquina de estado de "cerradura de combinación" con salidas Mealy en la tabla 7-14 de la página 582. La misma máquina de estado está especificada por el programa ABEL en la tabla 7-31, que utiliza instrucciones `WITH` para las salidas Mealy. Advierta que los corchetes de cierre toman el lugar de los signos de punto y coma que normalmente terminan las instrucciones de transición para los estados.

Basados en la descripción en palabras de la cerradura de combinación, no es posible realizar `UNLK` y `HINT` como salidas canalizadas, puesto que dependen del valor actual de X. Sin embargo, si volvemos a definir `UNLK` para que sea afirmada por todo el estado "sin cerradura" y `HINT` para que sea el valor siguiente real recomendado de X, podemos crear una nueva máquina con salidas canalizadas, como se muestra en la tabla 7-32. Note que utilizamos el operador de asignación temporizado para las salidas. Lo que es más importante, advierta que los valores de `UNLK` y `HINT` son diferentes en comparación con el ejemplo de Mealy, puesto que tienen que "adelantarse" un tic de reloj.

Debido al "adelanto", el diseño y la comprensión de las salidas canalizadas pueden ser más difíciles que en el caso de las salidas Mealy. En el ejemplo anterior, no tuvimos que modificar la instrucción del problema. La ventaja de las salidas canalizadas es que, puesto que se encuentran conectadas directamente a salidas registradas, son

Tabla 7-31
Máquina de estado con definiciones de salida Mealy integradas.

```
module SMEX4
title 'Combination-Lock State Machine'

" Input and output pins
CLOCK, X                        pin;
Q1..Q3                          pin istype 'reg';
UNLK, HINT                      pin istype 'com';

" Definitions
S       = [Q1,Q2,Q3];           " State variables
ZIP     = [ 0, 0, 0];           " State encodings
X0      = [ 0, 0, 1];
X01     = [ 0, 1, 0];
X011    = [ 0, 1, 1];
X0110   = [ 1, 0, 0];
X01101  = [ 1, 0, 1];
X011011 = [ 1, 1, 0];
X0110111 = [ 1, 1, 1];

state_diagram S

state ZIP:      IF X==0 THEN X0     WITH {UNLK = 0; HINT = 1}
                ELSE ZIP            WITH {UNLK = 0; HINT = 0}

state X0:       IF X==0 THEN X0     WITH {UNLK = 0; HINT = 0}
                ELSE X01            WITH {UNLK = 0; HINT = 1}

state X01:      IF X==0 THEN X0     WITH {UNLK = 0; HINT = 0}
                ELSE X011           WITH {UNLK = 0; HINT = 1}

state X011:     IF X==0 THEN X0110  WITH {UNLK = 0; HINT = 1}
                ELSE ZIP            WITH {UNLK = 0; HINT = 0}

state X0110:    IF X==0 THEN X0     WITH {UNLK = 0; HINT = 0}
                ELSE X01101         WITH {UNLK = 0; HINT = 1}

state X01101:   IF X==0 THEN X0     WITH {UNLK = 0; HINT = 0}
                ELSE X011011        WITH {UNLK = 0; HINT = 1}

state X011011:  IF X==0 THEN X0110  WITH {UNLK = 0; HINT = 0}
                ELSE X0110111       WITH {UNLK = 0; HINT = 1}

state X0110111: IF X==0 THEN X0     WITH {UNLK = 1; HINT = 1}
                ELSE ZIP            WITH {UNLK = 0; HINT = 0}

equations
S.CLK = CLOCK;

END SMEX4
```

válidas unos cuantos retardos de compuerta apenas después de un cambio de estado que las salida Moore o Mealy, las cuales normalmente incluyen lógica combinacional adicional. En el ejemplo de cerradura de combinación, probablemente no es tan importante abrir su cerradura o mirar su sugerencia unos cuantos nanosegundos antes. Sin embargo, cortar unos cuantos retardos de compuerta puede ser bastante importante en aplicaciones de alta velocidad.

Tabla 7-32
Máquina de estado con definiciones de salida canalizadas integradas.

```
module SMEX5
title 'Combination-Lock State Machine'

" Input and output pins
CLOCK, X                        pin;
Q1..Q3, UNLK, HINT              pin istype 'reg';

" Definitions
S        = [Q1,Q2,Q3];          " State variables
ZIP      = [ 0, 0, 0];          " State encodings
X0       = [ 0, 0, 1];
X01      = [ 0, 1, 0];
X011     = [ 0, 1, 1];
X0110    = [ 1, 0, 0];
X01101   = [ 1, 0, 1];
X011011  = [ 1, 1, 0];
X0110111 = [ 1, 1, 1];

state_diagram S

state ZIP:      IF X==0 THEN X0     WITH {UNLK := 0; HINT := 1}
                ELSE ZIP            WITH {UNLK := 0; HINT := 0}

state X0:       IF X==0 THEN X0     WITH {UNLK := 0; HINT := 1}
                ELSE X01            WITH {UNLK := 0; HINT := 1}

state X01:      IF X==0 THEN X0     WITH {UNLK := 0; HINT := 1}
                ELSE X011           WITH {UNLK := 0; HINT := 0}

state X011:     IF X==0 THEN X0110  WITH {UNLK := 0; HINT := 1}
                ELSE ZIP            WITH {UNLK := 0; HINT := 0}

state X0110:    IF X==0 THEN X0     WITH {UNLK := 0; HINT := 1}
                ELSE X01101         WITH {UNLK := 0; HINT := 1}

state X01101:   IF X==0 THEN X0     WITH {UNLK := 0; HINT := 1}
                ELSE X011011        WITH {UNLK := 0; HINT := 1}

state X011011:  IF X==0 THEN X0110  WITH {UNLK := 0; HINT := 1}
                ELSE X0110111       WITH {UNLK := 1; HINT := 0}

state X0110111: IF X==0 THEN X0     WITH {UNLK := 0; HINT := 1}
                ELSE ZIP            WITH {UNLK := 0; HINT := 0}

equations
S.CLK = CLOCK; UNLK.CLK = CLOCK; HINT.CLK = CLOCK;

END SMEX5
```

7.11.6 Vectores de prueba

Los vectores de prueba para circuitos secuenciales en ABEL tienen los mismos usos y limitaciones que los vectores de prueba para los circuitos combinacionales, como se describe en la sección 4.6.7. Una importante adición a su sintaxis es el uso de la constante ".C." para indicar un flanco de reloj, $0 \rightarrow 1 \rightarrow 0$. De este modo, la tabla 7-33 es un programa ABEL, con vectores de prueba, para un registro de 8 bits simple con una entrada de habilitación de reloj. Varios vectores se utilizan para probar diferentes valores de entrada de carga y retención.

flanco de reloj .C.

```
module REG8EN
title '8-bit register with clock enable'

" Input and output pins
CLK, EN, D1..D8         pin;
Q1..Q8                  pin istype 'reg';

" Sets
D = [D1..D8];
Q = [Q1..Q8];

equations

Q.CLK = CLK;

WHEN EN == 1 THEN Q := D ELSE Q := Q;

test_vectors ([CLK, EN,  D  ] -> [ Q  ])
              [.C.,  1, ^h00] -> [^h00]; " 0s in every bit
              [.C.,  0, ^hFF] -> [^h00]; " Hold capability, EN=0
              [.C.,  1, ^hFF] -> [^hFF]; " 1s in every bit
              [.C.,  0, ^h00] -> [^hFF]; " Hold capability
              [.C.,  1, ^h55] -> [^h55]; " Adjacent bits shorted
              [.C.,  0, ^hAA] -> [^h55]; " Hold capability
              [.C.,  1, ^hAA] -> [^hAA]; " Adjacent bits shorted
              [.C.,  1, ^h55] -> [^h55]; " Load with quick setup
              [.C.,  1, ^hAA] -> [^hAA]; " Again
END REG8EN
```

Tabla 7-33
Programa ABEL con vectores de prueba para un registro de 8 bits simple.

Un enfoque típico para probar máquinas de estado es escribir vectores que causen que la máquina no solamente visite cada estado, sino también ejercitar cada transición de cada estado. Una diferencia clave y desafío comparado con los vectores de prueba del circuito combinacional es que los vectores primero deben conducir a la máquina al estado deseado antes de probar una transición, y luego regresar de nueva cuenta para cada transición diferente de ese estado.

De este modo, la tabla 7-34 muestra vectores de prueba para la máquina de estado en la tabla 7-27. Es importante comprender que, a diferencia de los vectores combinacionales, estos vectores trabajan solamente si se aplican exactamente en el orden que son escritos. Note que los vectores fueron escritos para ser independientes de la codificación de estado. Como resultado, no tienen que ser modificados si la codificación del estado está cambiando.

Encontramos otro reto si intentamos crear vectores de estado para la máquina de estado de cerradura de combinación de la tabla 7-31 en la página 637. Esta máquina tiene un problema grave cuando llega a probarse: no tiene entrada de restablecimiento. Su estado de inicio en el arranque puede ser diferente en tecnologías y dispositivos basados en PLD: los flip-flops individuales pueden estar todos establecidos, todos restablecidos o todos en estados aleatorios. En la aplicación real de la máquina, no necesitamos necesariamente una entrada de restablecimiento, pero para propósitos de prueba de alguna manera tenemos que obtenerla para un estado de inicio conocido.

Tabla 7-34 Vectores de prueba para la máquina de estado en la tabla 7-27.

```
test_vectors
([RESET_L, CLOCK, A, B] -> [QSTATE , LASTA, Z])
 [   0   ,  .C. , 0, 0] -> [INIT    ,   0   , 0]; " Check -->INIT (RESET)
 [   0   ,  .C. , 1, 0] -> [INIT    ,   1   , 0]; "  and LASTA flip-flop
 [   1   ,  .C. , 0, 0] -> [LOOKING,   0   , 0]; " Come out of initialization
 [   0   ,  .C. , 0, 0] -> [INIT    ,   0   , 0]; " Check LOOKING-->INIT (RESET)
 [   1   ,  .C. , 0, 0] -> [LOOKING,   0   , 0]; " Come out of initialization
 [   1   ,  .C. , 1, 0] -> [LOOKING,   1   , 0]; " --> LOOKING since 0!=1
 [   1   ,  .C. , 1, 0] -> [OK      ,   1   , 1]; " --> OK since 1==1
 [   0   ,  .C. , 0, 0] -> [INIT    ,   0   , 0]; " Check OK-->INIT (RESET)
 [   1   ,  .C. , 0, 0] -> [LOOKING,   0   , 0]; " Go back towards OK ...
 [   1   ,  .C. , 0, 0] -> [OK      ,   0   , 1]; " --> OK since 0==0
 [   1   ,  .C. , 1, 1] -> [OK      ,   1   , 1]; " --> OK since B, even though 1!=0
 [   1   ,  .C. , 1, 0] -> [OK      ,   1   , 1]; " --> OK since 1==1
 [   1   ,  .C. , 0, 0] -> [LOOKING,   0   , 0]; " --> LOOKING since 0!=1
```

secuencia de sincronización

Afortunadamente, la máquina de cerradura de combinación tiene una *secuencia de sincronización*: una secuencia fija de uno o más valores de entrada que siempre la conduce a un cierto estado conocido. En particular, comenzando de cualquier estado, si aplicamos X = 1 a la máquina para cuatro tics, siempre estaremos en el estado ZIP por el cuarto tic. Éste es el enfoque tomado por los primeros cuatro vectores en la tabla 7-35. Hasta que obtenemos el estado conocido, indicamos el estado siguiente en el lado derecho del vector como si fuera "sin importancia", de modo que el simulador o probador del dispositivo físico no señale un estado aleatorio como un error.

Una vez que ya estamos encaminados, encontramos otra cosa que es nueva: las salidas Mealy deben probarse. Como mostramos mediante el cuarto y quinto vectores, no tenemos que provocar la transición del reloj en cada vector de prueba. En cambio, podemos mantener el reloj fijo en 0, donde la última transición lo deja, y observar los valores de salida Mealy producidos por los dos valores de entrada de X. Podemos entonces probar la transición del estado siguiente.

Para las transiciones de estado, enumeramos el estado siguiente esperado pero mostramos los valores de salida como "sin importancia". Para un vector de prueba correcto las salidas deben mostrar los valores alcanzados *después* de la transición, una función del estado *siguiente*. Aunque es posible averiguarlos e incluirlos, la complejidad es suficiente para que usted consiga un buen dolor de cabeza, y de todas maneras serán probados por los vectores CLOCK=0 siguientes.

La creación de vectores de prueba para una máquina de estado a mano es un proceso minucioso, y no importa cuánto esmero tenga usted, no hay garantía de que se hayan

SINCRONIZACIÓN DE SECUENCIAS Y ENTRADAS DE RESTABLECIMIENTO

Fuimos afortunados con la cerradura de combinación; no todas las máquinas de estado tienen secuencias de sincronización. Debido a esto la mayor parte las máquinas de estado se diseñan con una entrada de restablecimiento, la que en efecto, permite obtener una secuencia de sincronización de longitud uno.

Tabla 7-35 Vectores de prueba para la máquina de estado de cerradura de combinación de la tabla 7-31.

```
test_vectors
([CLOCK, X] -> [   S    , UNLK, HINT])
 [ .C. , 1] -> [.X.     , .X. , .X. ]; " Since no reset input, apply
 [ .C. , 1] -> [.X.     , .X. , .X. ]; "   a 'synchronizing sequence'
 [ .C. , 1] -> [.X.     , .X. , .X. ]; "   to reach a known starting
 [ .C. , 1] -> [ZIP     , .X. , .X. ]; "   state
 [  0  , 0] -> [ZIP     ,  0  ,  1  ]; " Test Mealy outputs for both
 [  0  , 1] -> [ZIP     ,  0  ,  0  ]; "   values of X
 [ .C. , 1] -> [ZIP     , .X. , .X. ]; " Test ZIP-->ZIP (X==1)
 [ .C. , 0] -> [X0      , .X. , .X. ]; "   and ZIP-->X0 (X==0)
 [  0  , 0] -> [X0      ,  0  ,  0  ]; " Test Mealy outputs for both
 [  0  , 1] -> [X0      ,  0  ,  1  ]; "   values of X
 [ .C. , 0] -> [X0      , .X. , .X. ]; " Test X0-->X0 (X==0)
 [ .C. , 1] -> [X01     , .X. , .X. ]; "   and X0-->X01 (X==1)
 [  0  , 0] -> [X01     ,  0  ,  0  ]; " Test Mealy outputs for both
 [  0  , 1] -> [X01     ,  0  ,  1  ]; "   values of X
 [ .C. , 0] -> [X0      , .X. , .X. ]; " Test X01-->X0 (X==0)
 [ .C. , 1] -> [X01     , .X. , .X. ]; " Get back to X01
 [ .C. , 1] -> [X011    , .X. , .X. ]; " Test X01-->X011 (X==1)
```

probado todas sus fallas potenciales de hardware y de funciones. Por ejemplo, los vectores en la tabla 7-34 no prueban (A LASTA) = 10 en el estado LOOKING, o (AB LASTA) = 100 en el estado OK. De este modo, la generación de un conjunto completo de vectores de prueba con la finalidad de detectar fallas es un proceso que se deja mejor para un programa automático de generación de pruebas. En la tabla 7-35 nos agotamos después de escribir los vectores para los primeros estados; se deja como un ejercicio (7.92) completar los vectores de prueba. Con todo, por el lado de prueba funcional, escribir unos cuantos vectores para ejercitar las funciones más básicas de la máquina puede desbrozar errores de diseño obvios al principio del proceso. Los errores de diseño más sutiles se detectan mejor mediante una minuciosa simulación al nivel del sistema.

7.12 Características de diseño de circuito secuencial VHDL

La mayoría de las características de VHDL que son necesarias para apoyar el diseño de circuitos secuenciales, en particular, los procesos, ya fueron introducidas en la sección 4.7 y se utilizaron en las secciones VHDL en el capítulo 5. Esta sección introduce justamente un par más de características y proporciona ejemplos simples de cómo se utilizan. Los ejemplos más extensos aparecen en las secciones VHDL del capítulo 8.

7.12.1 Circuitos secuenciales de retroalimentación

Un proceso VHDL y el mecanismo de lista de eventos del simulador para rastrear cambios de señal forman la base fundamental para manejar circuitos secuenciales de retroalimentación en VHDL. Recuerde que los circuitos secuenciales de retroalimentación pueden cam-

Tabla 7-36
Flujo de datos VHDL para un latch S-R.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vsrlatch is
  port (S, R: in STD_LOGIC;
        Q, QN: buffer STD_LOGIC );
end Vsrlatch;

architecture Vsrlatch_arch of Vsrlatch is
begin
  QN <= S nor Q;
  Q  <= R nor QN;
end Vsrlatch_arch;
```

biar de estado en respuesta a cambios de entrada, y que éstos se manifiestan mediante cambios que se propagan en un lazo de retroalimentación hasta que se estabiliza el lazo de retroalimentación. En simulación, esto se manifiesta por el simulador colocando cambios de señal en la lista de eventos y procesos de programación para volver a ejecutar en "tiempo delta" y propagar estos cambios de señal hasta que no se programen más cambios de señal.

La tabla 7-36 es un programa VHDL para un latch S-R. La arquitectura contiene dos instrucciones de asignación concurrentes, cada una de las cuales da origen a un proceso, como se discutió en la sección 4.7.9. Estos procesos interactúan para crear el comportamiento de cierre simple de un latch S-R.

La simulación VHDL es suficientemente fiel para manejar el caso donde tanto S como R se afirman simultáneamente. El resultado más interesante en simulación se presenta si se niegan S y R de manera simultánea. Recuerde del cuadro de la página 536 que un latch S-R real puede oscilar o irse a un estado metaestable en esta situación. La simulación potencialmente se reciclará permanentemente a medida que cada ejecución de una instrucción de asignación dispare una ejecución de la otra. Después de algún número de repeticiones, un simulador bien diseñado descubrirá el problema (el tiempo delta se mantiene avanzando mientras que el tiempo simulado, no) y detiene la simulación.

¿QUÉ ES LO QUE 'U' QUIERES?

Sería bueno si el modelo de latch S-R en la tabla 7-36 produjera una salida 'U' siempre que S y R fueran negadas simultáneamente, pero no es tan bueno. Sin embargo, el lenguaje es lo suficientemente poderoso para que los diseñadores VHDL con experiencia puedan fácilmente escribir un modelo con ese comportamiento. Un modelo de esa clase haría uso de las facilidades de modelado de tiempo de VHDL las cuales no hemos discutido, para modelar el "tiempo de recuperación" del latch (véase el cuadro en la página 537) y obligar una salida 'U' si ocurre un segundo cambio de entrada demasiado pronto. Incluso es posible modelar un tiempo de resolución de metaestabilidad máximo supuesto de esta forma.

Advierta que si un circuito tiene la posibilidad de entrar a un estado metaestable, no hay garantía de que la simulación lo detectará, especialmente en diseños más grandes. La mejor manera de evitar los problemas de metaestabilidad en el diseño de un sistema es identificar claramente y proteger sus fronteras asincrónicas, como se discute en la sección 8.9.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity VposDff is
  port (CLK, CLR, D: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end VposDff;

architecture VposDff_arch of VposDff is
begin
  process (CLK, CLR)
  begin
    if CLR='1' then Q <= '0'; QN <= '1';
    elsif CLK'event and CLK='1' then Q <= D; QN <= not D;
    end if;
  end process;
end VposDff_arch;
```

Tabla 7-37
VHDL de comportamiento para un flip-flop D disparado por flanco positivo.

7.12.2 Circuitos temporizados

En la práctica, la mayoría de los diseños digitales que se modelan mediante el lenguaje VHDL son sistemas síncronos, temporizados que emplean flip-flops disparados por flanco. Además de lo que ya hemos aprendido acerca de VHDL, hay una característica más que es necesaria para describir el comportamiento disparado por el flanco. El *atributo* `event` puede ser vinculado a un nombre de señal para producir un valor de tipo `boolean` que es `true` (verdadero) si un evento en la señal provocó que el proceso involucrado se ejecutara en el ciclo de simulación actual, y `false` en caso contrario.

atributo event

Utilizando el atributo `event`, podemos modelar el comportamiento de un flip-flop D disparado por flanco positivo con borrado asincrónico como se muestra en la tabla 7-37. Aquí, la entrada de borrado asincrónico CLR cancela cualquier comportamiento en la entrada de reloj CLK y por lo tanto se verifica primero, en la cláusula "`if`". Si CLR es negada, entonces se verifica la cláusula "`elsif`" y sus instrucciones se ejecutan en el flanco ascendente de CLK. Advierta que "`CLK' event`" es verdadero para cualquier cambio en CLK y "CLK= '1'" se verifica para limitar el disparo sólo al flanco ascendente de CLK. Existen muchas otras maneras para construir procesos o instrucciones con comportamiento de disparo por flanco; la tabla 7-38 muestra dos formas adicionales para describir un flip-flop D (sin una entrada CLR).

En el banco de pruebas para circuito temporizado, una de las cosas que necesitan hacerse en generar una señal de reloj del sistema. Esto puede hacerse con bastante facilidad mediante un ciclo dentro de un proceso, como se muestra en la tabla 7-39 para un reloj de 100 MHz con un ciclo de trabajo del 60%.

```
process
  wait until CLK'event and CLK='1';
  Q <= D;
end process;

Q <= D when CLK'event and CLK='1' else Q;
```

Tabla 7-38
Dos maneras adicionales para describir un flip-flop D disparado por flanco positivo.

MATERIA DE SÍNTESIS

Usted podrá preguntarse ¿cómo convierte una herramienta de síntesis el comportamiento disparado por flanco descrito en la tabla 7-37 o 7-38 en una eficiente realización de flip-flop? La mayoría de las herramientas reconocen solamente unas cuantas maneras predeterminadas de expresar comportamiento disparado por flanco y mapear aquellas en componentes de flip-flop predeterminados en la tecnología objetivo.

El motor de síntesis de Sinopsis empleado en el software Xilinx Foundation Series 1.5 reconoce la expresión "`CLK' event and CLK= '1'`" que utilizamos en este libro. Incluso con eso como algo dado, VHDL tiene muchas formas diferentes de expresar la misma funcionalidad, como mostramos en la tabla 7-38. Peter Ashenden, autor de *The Designer's Guide to VHDL* (Morgan Kaufmann, 1996), procesó estas instrucciones y alguna otra, con alguna modificación, a través de varias herramientas de síntesis diferentes. Solamente una de ellas fue capaz de sintetizar tres de las cuatro formas; la mayoría podía manejar solamente dos. Así, usted necesita seguir el método que está prescrito por la herramienta que se utiliza.

Tabla 7-39
Proceso de reloj dentro de un banco de pruebas.

```
architecture TB_arch of TB is
signal MCLK: STD_LOGIC;
signal ... -- Declare otras señales de entrada y salida

process -- Generador de reloj
begin
  MCLK <= '1';  -- Inicie en 1 al tiempo 0
  loop
    MCLK <= '0' after 6 ns;
    MCLK <= '1' after 4 ns;
  end loop;
end process;

process -- Genere el resto de los estímulos de entrada,
          verifique las salidas.
begin
  ...
end;
```

Referencias

El problema de la metaestabilidad ha estado presente por mucho tiempo. Los filósofos griegos escribieron acerca del problema de la indecisión hace ya miles de años. Un grupo de filósofos modernos llamados Devo cantaron acerca de la metaestabilidad en la canción principal de su álbum *Freedom of Choice*. Y el congreso de los Estados Unidos de Norteamérica todavía no puede decidir cómo "ahorrar" el Seguro Social.

La capacidad de rastreo o exploración fue utilizada primero en latchs, no en flip-flops, en diseños de CI de IBM hace ya décadas. Edward J. McCluskey tiene una discusión muy buena de éste y otros métodos de exploración en *Logic Design Principles* (Prentice Hall, 1986).

La mayoría de los diseños ASIC basados en MSI, PLD y FPGA hacen uso de los tipos de circuito secuencial descritos en este capítulo. Sin embargo, existen otros tipos que se utilizan tanto en diseños discretos antiguos (pasando por todo el camino de vuelta hasta la lógica de tubos de vacío) como también en diseños VLSI personalizados modernos.

Por ejemplo, las máquinas de estado sincrónicas temporizadas son un caso especial de una clase más general de *circuitos en modo de pulso*. Tales circuitos tienen una o más *entradas de pulsos* tales que a) solamente se presenta un pulso a un tiempo; b) las entradas sin pulso son estables cuando se presenta un pulso; c) solamente los pulsos pueden ocasionar cambios de estado, y d) un pulso provoca a lo más un cambio de estado. En máquinas de estado sincrónicas temporizadas, el reloj es la entrada de pulso simple, y un "pulso" es el flanco de disparo del reloj. Sin embargo, también es posible construir circuitos con múltiples entradas de pulsos, y es posible emplear otros elementos de almacenamiento aparte de los familiares flip-flops disparados por flanco. Estas posibilidades se discuten muy ampliamente en los *Logic Design Principles* de McCluskey (Prentice Hall, 1986).

circuito en modo de pulso
entrada de pulso

Un tipo particularmente importante de circuito en modo de pulso que se discute por McCluskey y otros autores es la *máquina de latch de dos fases*. La razón para un enfoque de temporización de dos fases en circuitos VLSI se discute por Carver Mead y Lynn Conway en *Introduction to VLSI Systems* (Addison-Wesley, 1980). Estas máquinas esencialmente eliminan los riesgos esenciales presentes en flip-flops disparados por flanco empleando pares de latchs que son habilitados por relojes que no se traslapan.

máquina de latch de dos fases

Los métodos para reducir tanto tablas de estado completa e incompletamente especificadas se describen en textos de diseño de lógica avanzados, incluyendo el libro de 1986 de McCluskey. Una discusión más matemática de estos métodos y otros temas "teóricos" en el diseño de máquina secuencial aparece en *Switching and Finite Automata Theory*, segunda edición, por Zvi Kohavi (McGraw-Hill, 1978).

Como mostramos en este capítulo, los diagramas de estado inadecuadamente construidos pueden conducir a una descripción ambigua del comportamiento del estado siguiente. Las estructuras "`IF-THEN-ELSE`" en HDL como ABEL y VHDL pueden eliminar estas ambigüedades, pero no fueron las primeras en hacerlo así. La notación de *máquina de estado algorítmica* (*ASM, algorithmic-state machine*), un equivalente tipo diagrama de flujo de las instrucciones anidadas `IF-THEN-ELSE`, ha estado con nosotros por cerca de 25 años. Los denominados *diagramas ASM* fueron aplicados primero en los laboratorios Hewlett-Packard por Thomas E. Osborne y fueron desarrollados adicionalmente por su colega Christopher R. Clare en el libro, *Designing Logic Systems Using State Machines* (McGraw-Hill, 1973). Los métodos de diseño y síntesis que usan diagramas ASM encontraron posteriormente un lugar en muchos textos de diseño digital, incluyendo *The Art of Digital Design* por F. P. Prosser y D. E. Winkel (Prentice Hall, 1987, segunda edición) y *Digital Desing* de M. Morris Mano (Prentice Hall, 1984), además de las primeras dos ediciones del libro que usted está leyendo.

máquina de estado algorítmica
esquema ASM

Otra notación para describir máquinas de estado es una extensión de la notación de diagrama de estado "tradicional" llamado diagrama de estado documentado mnemónico (MDS, *mnemonic documented state*); fue desarrollada por William I. Fletcher en *An Engineering Approach to Digital Design* (Prentice Hall, 1980). Los diagramas ASM y los diagramas MDS actualmente han sido reemplazados en gran parte por HDL y sus compiladores.

Muchos ambientes CAD para diseño digital incluyen una herramienta de entrada de diagrama de estado gráfica. Desgraciadamente, éstos por lo regular apoyan solamente los diagramas de estado tradicionales, facilitando al diseñador la creación de una descripción ambigua del comportamiento del estado siguiente. Como resultado, mi recomendación personal es que usted se mantenga alejado de los editores de diagramas de estado y en su lugar utilice un HDL para describir sus máquinas de estado.

Mencionamos la importancia de sincronizar secuencias en conexión con los vectores de prueba de máquina de estado. Existe en la actualidad una teoría y práctica muy bien desarrolladas pero casi olvidadas de sincronización de secuencias y "experimentos caseros" algo menos impactantes, descritos por Frederick C. Hennie en *Finite-State Models for Logical Machines* (Wiley, 1968). A menos que usted tenga este viejo clásico en su librero y sepa cómo aplicar sus enseñanzas, por favor, ¡sólo recuerde proporcionar una entrada de restablecimiento en cada máquina de estado que usted diseñe!

Problemas propuestos

7.1 Proporciona tres ejemplos de metaestabilidad que ocurren en la vida diaria, aparte de los discutidos en este capítulo.

7.2 Dibuje las salidas de un latch S-R del tipo mostrado en la figura 7-5 para las formas de onda de entrada mostradas en la figura X7.2. Suponga que los tiempos de ascenso y caída de entrada y salida son cero, que el retardo de propagación de una compuerta NOR es 10 ns y que cada división de tiempo es de 10 ns.

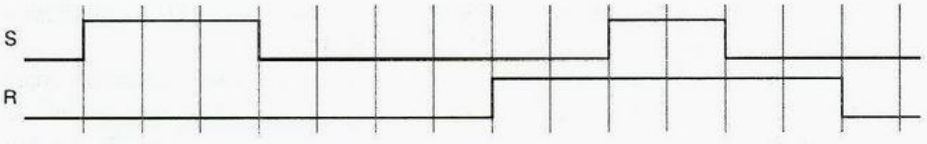


Figura X7.2

7.3 Repita el problema 7.2 utilizando las formas de onda de entrada mostradas en la figura X7.3. Aunque usted puede encontrar increíble el resultado, este comportamiento puede ocurrir en realidad en dispositivos reales cuyos tiempos de transición sean cortos en comparación con sus retardos de propagación.

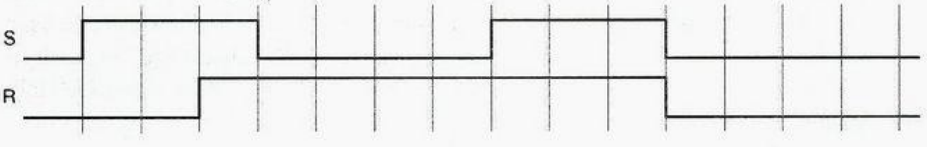


Figura X7.3

7.4 La figura 7-34 mostraba cómo construir un flip-flop T con habilitación utilizando un flip-flop D y lógica combinacional. Muestre cómo construir un flip-flop D empleando un flip-flop T con lógica combinacional y habilitación.

7.5 Muestre cómo construir un flip-flop J-K con un flip-flop T con lógica combinacional y habilitación.

7.6 Muestre cómo construir un latch S-R que utiliza un flip-flop D disparado por flanco positivo 74x74 y *ningún* otro componente.

7.7 Muestre cómo construir un flip-flop equivalente al flip-flop J-$\overline{K}$ disparado por flanco positivo que utiliza un flip-flop D disparado por flanco positivo 74x74 y una o más compuertas de un encapsulado 74x00.

7.8 Muestre cómo construir un flip-flop equivalente al flip-flop D disparado por flanco positivo 74x74 empleando un flip-flop J-$\overline{K}$ disparado por flanco positivo 74x109 y *ningún* otro componente.

7.9 Analice la máquina de estado sincrónica temporizada en la figura X7.9. Escriba las ecuaciones de excitación, la tabla de excitación/transición y la tabla de estado/salida (utilice los nombres de estado A-D para Q1 Q2 = 00-11).

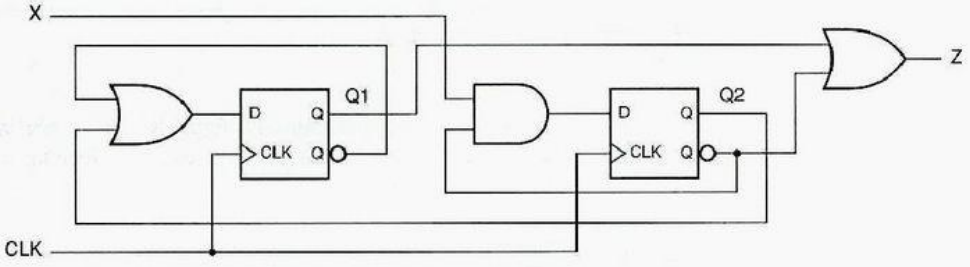


Figura X7.9

7.10 Repita el problema 7.9, intercambiando compuertas AND y OR en el diagrama lógico. ¿Es la nueva tabla de estado/salida el "doble" del original? Explique por qué.

7.11 Dibuje un diagrama de estado para la máquina de estado descrita por la tabla 7-6.

7.12 Dibuje un diagrama de estado para la máquina de estado descrita por la tabla 7-12.

7.13 Dibuje un diagrama de estado para la máquina de estado descrita por la tabla 7-14.

7.14 Construya una tabla de salida y estado equivalente al diagrama de estado en la figura X7.14. Advierta que el diagrama está dibujado con la convención de que el estado no cambia excepto para las condiciones de entrada que se muestran de manera explícita.

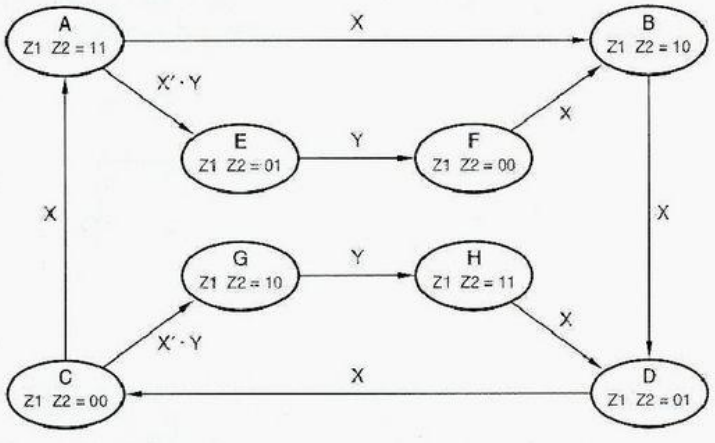
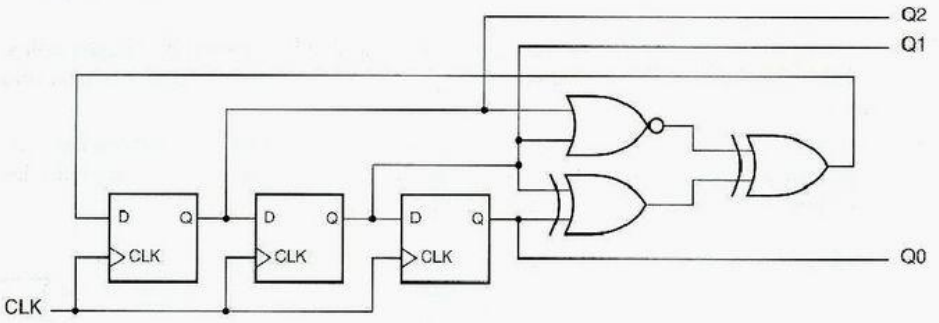


Figura X7.14

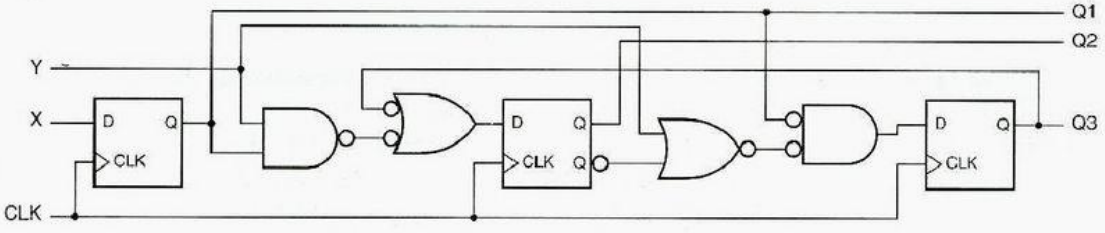
7.15 Analice la máquina de estado sincrónica temporizada en la figura X7.15. Escriba las ecuaciones de excitación, tabla de excitación/transición y tabla de estado (utilice los nombres A-H para Q2 Q1 Q0 = 000-111).

Figura X7.15



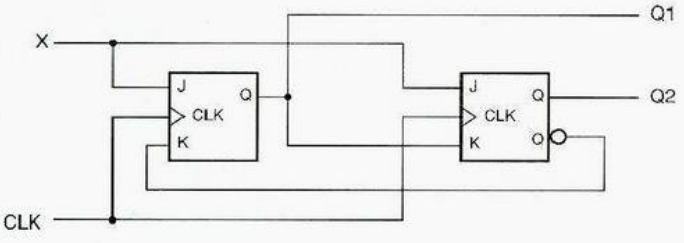
7.16 Analice la máquina de estado sincrónica temporizada en la figura X7.16. Escriba las ecuaciones de excitación, tabla de excitación/transición y tabla de estado (utilice los nombres de estado A-H para Q1 Q2 Q3 = 000-111).

Figura X7.16



7.17 Analice la máquina de estado sincrónica temporizada en la figura X7.17. Escriba las ecuaciones de excitación, ecuaciones de transición, tabla de transición y tabla de estado/salida (utilice los nombres de estado A-D para Q1 Q2 = 00-11). Dibuje un diagrama de estado y trace un diagrama de temporización para CLK, X, Q1 y Q2 para 20 tics de reloj, suponiendo que la máquina comienza en el estado 00 y X es continuamente 1.

Figura X7.17



7.18 Analice la máquina de estado sincrónica temporizada en la figura X7.18. Escriba las ecuaciones de excitación, ecuaciones de transición, tabla de transición y tabla de estado/salida (utilice los nombres A-D para Q1 Q0 = 00=11). Dibuje un diagrama de estado y trace un diagrama de temporización para CLK, EN, Q1 y Q0 para 10 tics de reloj, suponiendo que la máquina comienza en el estado 00 y EN es continuamente 1.

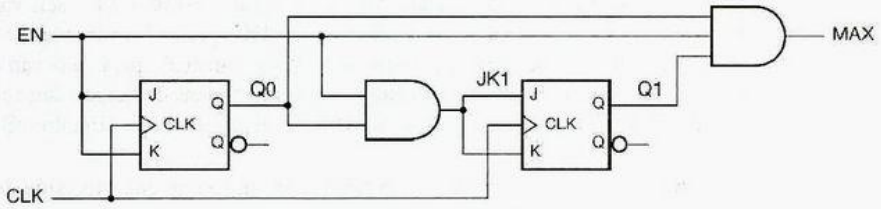


Figura X7.18

7.19 Analice la máquina de estado sincrónica temporizada en la figura X7.19. Escriba las ecuaciones de excitación, la tabla de excitación/transición y la tabla de estado/salida (utilice los nombres A-D para Q1 Q2 = 00-11).

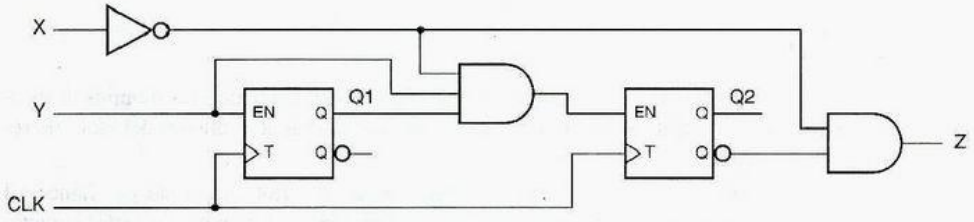
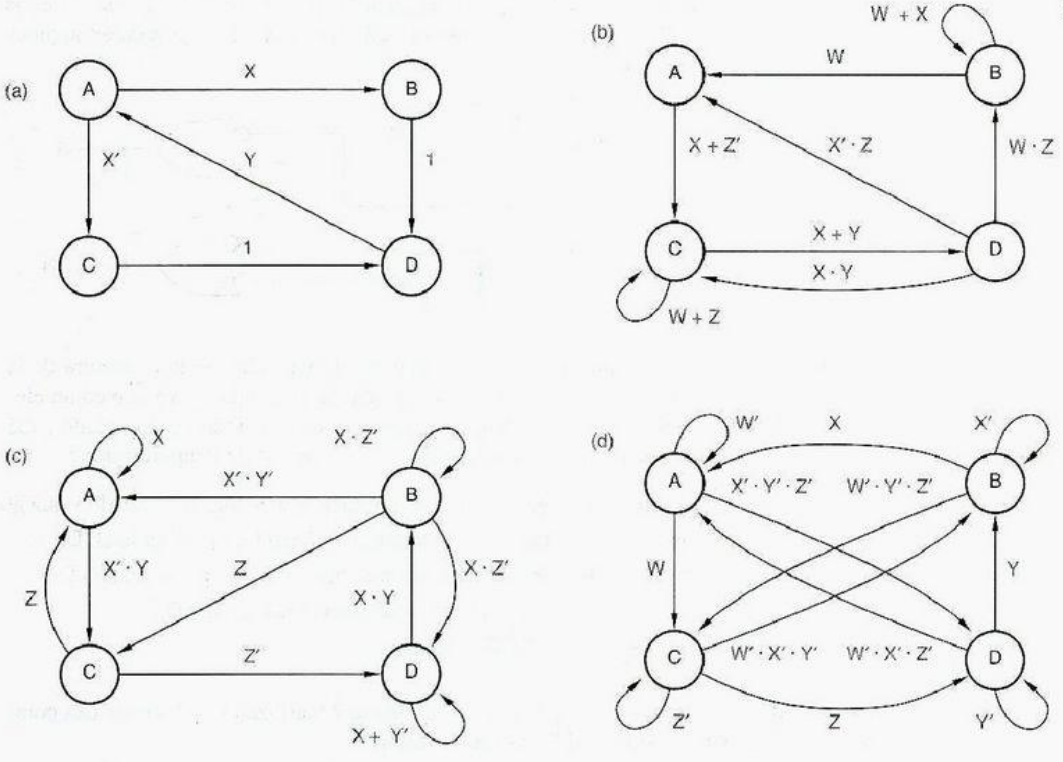


Figura X7.19

7.20 Todos los diagramas de estado en la figura X7.20 son ambiguos. Enumere todas las ambigüedades en estos diagramas de estado. (*Sugerencia:* Utilice mapas de Karnaugh donde sea necesario para hallar combinaciones de entrada sin cubrir o doblemente cubiertas.)

Figura X7.20



7.21 Sintetice un circuito para el diagrama de estado de la figura 7-64 utilizando seis variables para codificar el estado, donde las salidas LA-LC y RA-RC igualan las variables de estado mismas. Escriba una lista de transición, una ecuación de transición para cada variable de estado como una suma de p términos y ecuaciones simplificadas de transición/excitación para una realización empleando flip-flops D. Dibuje un diagrama de circuito utilizando componentes SSI y MSI.

7.22 Comenzando con la lista de transición en la tabla 7-18, encuentre una expresión de suma de productos mínima para Q2*, suponiendo que los estados siguientes para los estados sin utilizar sean verdaderos sin importancia.

7.23 Modifique el diagrama de estado de la figura 7-64 de modo que la máquina vaya al modo de riesgo inmediatamente si LEFT y RIGHT son afirmados simultáneamente durante una vuelta. Escriba la lista de transición correspondiente.

Ejercicios

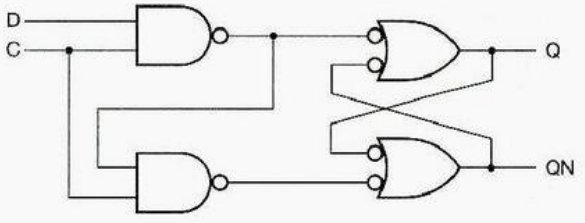
7.24 Explique cómo se presenta la metaestabilidad en un latch D cuando los tiempos de establecimiento y de retención no coinciden, analizando el comportamiento del ciclo de retroalimentación dentro del latch.

7.25 ¿Cuál es el tiempo de establecimiento mínimo de un flip-flop disparado por flanco tal como un flip-flop S-R o J-K maestro/esclavo? (*Sugerencia:* Depende de ciertas características del reloj.)

7.26 Describa una situación, aparte del estado metaestable, en la cual las salidas Q y QN de un flip-flop D disparado por flanco 74x74 pueden ser del tipo no complementarias durante un tiempo arbitrariamente largo.

7.27 Compare el circuito en la figura X7.27 con el latch D en la figura 7-12. Pruebe que los circuitos funcionan idénticamente. ¿En qué forma la figura X7.27, utilizada en algunos latchs D comerciales, es mejor?

Figura X7.27



7.28 Supongamos que una máquina de estado sincrónica temporizada con la estructura de la figura 7-35 se diseña empleando latchs D con entradas C de estado activo alto como elementos de almacenamiento. Para un funcionamiento de estado siguiente apropiado ¿qué relaciones deben ser satisfechas entre los siguientes parámetros de temporización?

t_{Fmin}, t_{Fmax}	Retardo de propagación mínimo y máximo de la lógica de estado siguiente.
t_{CQmin}, t_{CQmax}	Retardo de reloj hasta la salida mínimo y máximo para un latch D.
t_{DQmin}, t_{DQmax}	Retardo de datos hasta la salida mínimo y máximo para un latch D.
$tt_{establecimiento}$, $tt_{retención}$	Tiempos de establecimiento y retención para un latch D.
t_H, t_L	Tiempos de reloj ALTO y BAJO.

7.29 Vuelva a diseñar la máquina de estado en el problema 7.9 utilizando solamente tres compuertas de inversión (NAND o NOR) y no inversores.

7.30 Dibuje un diagrama de estado para una máquina de estado sincrónica temporizada con dos entradas, INIT y X, y una salida Z tipo Moore. Mientras que INIT es afirmada, Z es continuamente 0. Una vez que INIT es negada, Z debería permanecer como 0 hasta que X haya sido 0 durante dos tics sucesivos y 1 durante dos tics sucesivos, sin tener en cuenta el orden de la incidencia. Entonces Z debería ir a 1 y permanecer en 1 hasta que INIT sea afirmada de nueva cuenta. Su diagrama de estado debería ser cuidadosamente dibujado y planar (sin líneas cruzadas). (*Sugerencia:* No se requieren más de diez estados.)

7.31 Repita el ejercicio 7.30, pero escriba el diagrama de estado en ABEL.

7.32 Diseñe una máquina de estado sincrónica temporizada que verifique la paridad par de una línea de datos serial. El circuito debería tener dos entradas, SYNC y DATA, además de CLOCK, y una salida tipo Moore, ERROR. Invente una tabla de estado/salida que haga el trabajo y utilice solamente cuatro estados e incluya una descripción del significado de cada estado en la tabla. Elija una asignación de estado de 2 bits, escriba las ecuaciones de transición y de excitación y dibuje el diagrama lógico. Su circuito puede utilizar flip-flops D, flip-flops J-K, o uno de cada uno.

7.33 Repita el ejercicio 7.32, pero haga el diseño empleando ABEL y un PLD GAL16V8.

7.34 Diseñe una máquina de estado síncrona temporizada con la tabla de estado/salida mostrada en la tabla X7.34, utilice flip-flops D. Haga uso de dos variables de estado, Q1 Q2, con la asignación de estado A = 00, B = 01, C = 11, D = 10.

Tabla X7.34

	X		
S	0	1	Z
A	B	D	0
B	C	B	0
C	B	A	1
D	B	C	0
	S*		

7.35 Repita el ejercicio 7.34 empleando flip-flops J-K.

7.36 Escriba una nueva tabla de transición y derive ecuaciones de salida y excitación de costo mínimo para la tabla de estado en la tabla 7-6 empleando la asignación de estado "más simple" en la tabla 7-7 y flip-flops D. Compare el costo de su lógica de salida y excitación (cuando se realiza con un circuito AND-OR de dos niveles) con el circuito en la figura 7-54.

7.37 Repita el ejercicio 7.36 utilizando la asignación de estado "casi activo uno" en la tabla 7-7.

7.38 Supongamos que la máquina de estado en la figura 7-54 va a construirse utilizando flip-flops D 74LS74. ¿Qué señales deberían aplicarse a las entradas de prestablecimiento y borrado del flip-flop?

7.39 Escriba nuevas tablas de transición y excitación y derive las ecuaciones de salida y excitación de costo mínimo para la tabla de estado en la tabla 7-6 empleando la asignación de estado "más simple" en la tabla 7-7 y flip-flops J-K. Compare el costo de su lógica de salida y excitación (cuando se realiza con un circuito AND-OR de dos niveles) con el circuito en la figura 7-56.

7.40 Repita el ejercicio 7.39 aplicando la asignación de estado "casi activo uno" en la tabla 7-7.

7.41 Construya una tabla de aplicación similar a la tabla 7-10 para cada uno de los siguientes tipos de flip-flop: a) S-R; b) T con habilitación; c) D con habilitación. Discuta el problema único que usted encuentra cuando intenta hacer el uso más eficiente de elementos sin importancia con uno de estos flip-flops.

7.42 Construya una nueva tabla de excitación y derive las ecuaciones de salida y excitación de costo mínimo para la máquina de estado de la tabla 7-8 que utiliza flip-flops T con entradas de habilitación (figura 7-33). Compare el costo de su lógica de salida y excitación (cuando se realiza con un circuito AND-OR de dos niveles) con el circuito en la figura 7-54.

7.43 Determine la tabla completa de 8 estados del circuito en la figura 7-54. Utilice los nombres U1, U2 y U3 para los estados sin utilizar (001, 010 y 011). Dibuje un diagrama de estado y explique el comportamiento de los estados sin utilizar.

7.44 Repita el ejercicio 7.43 para el circuito de la figura 7-56.

7.45 Escriba una tabla de transición para la tabla de estado no mínima en la figura 7-51(a) que resulta de asignar los estados en orden de conteo binario, INIT-OKA1 = 000-110. Escriba las ecuaciones de excitación correspondientes para flip-flops D, suponiendo una disposición de costo mínimo del estado sin utilizar 111. Compare el costo de sus ecuaciones con las ecuaciones de costo mínimo para la tabla de estado mínima presentada en el texto.

7.46 Escriba la tabla de aplicación para un flip-flop T con habilitación.

7.47 En muchas aplicaciones, las salidas producidas por una máquina de estado durante o poco después del restablecimiento son irrelevantes, mientras que la máquina comienza a comportarse correctamente un breve tiempo después que la señal de restablecimiento es eliminada. Si se aplica esta idea a la tabla 7-6, el estado INIT puede eliminarse y solamente se necesitan dos variables de estado para codificar los cuatro estados restantes. Vuelva a diseñar la máquina de estado aplicando esta idea. Escriba una tabla de estado, tabla de transición, tabla de excitación para flip-flops D, ecuaciones de salida y excitación de costo mínimo y diagrama lógico nuevos. Compare el costo del nuevo circuito con el de la figura 7-54.

7.48 Repita el ejercicio 7-47 utilizando flip-flops J-K y emplee la figura 7-56 para comparar costo.

7.49 Vuelva a diseñar la máquina de conteo de unos de la tabla 7-12, asignando los estados en orden de conteo binario (S0-S3 = 00, 01, 10, 11). Compare el costo de las ecuaciones de excitación de suma de productos resultante con las derivadas en el texto.

7.50 Repita el ejercicio 7-49 utilizando flip-flops J-K.

7.51 Repita el ejercicio 7-49 utilizando flip-flops T con habilitación.

7.52 Vuelva a diseñar la máquina de conteo de unos de la tabla 7-12, como un diagrama de estado ABEL. Intente hallar una asignación de estado que minimice el número total de términos de producto, suponiendo que se pueda emplear cualquier polaridad de las ecuaciones de salida. ¿Cuántas asignaciones de estado diferentes deben examinarse?

7.53 Vuelva a diseñar la máquina de cerradura de combinación de la tabla 7-14, asignando estados codificados en orden de código de Gray (A-H = 000, 001, 011, 010, 110, 111, 101, 100). Compare el costo de las ecuaciones de excitación de suma de productos resultante con las derivadas en el texto.

7.54 Encuentre una asignación de estado de 3 bits para la máquina de cerradura de combinación de la tabla 7-14 que resulte en ecuaciones de excitación menos costosas que las derivadas en el texto. (*Sugerencia:* Utilice el hecho de que las entradas 1-3 son las mismas que las entradas 4-6 en la secuencia de entrada requerida.)

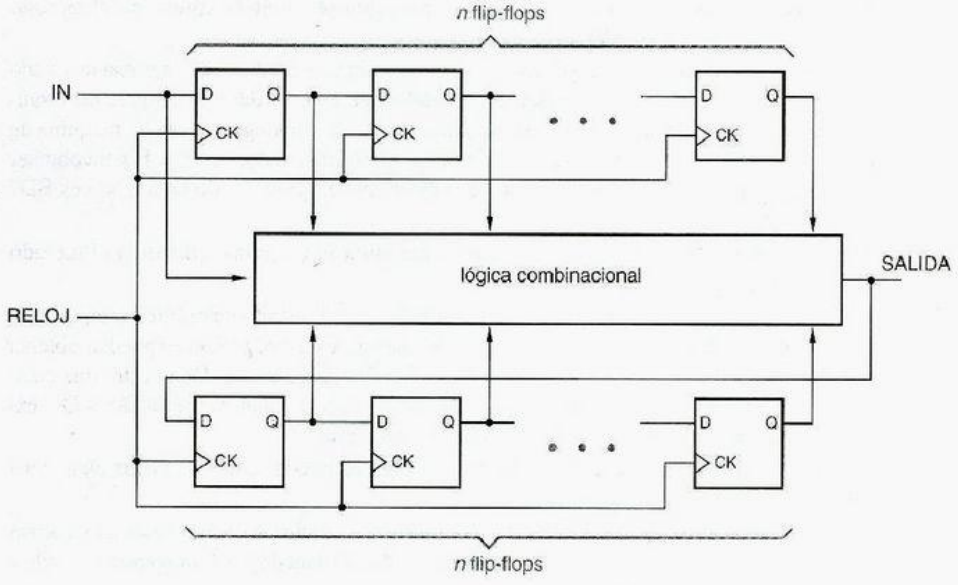


Figura X7.56

7.55 ¿Qué cambios serían hechos a las ecuaciones de salida y excitación para la máquina de cerradura de combinación en la sección 7.4.6 como resultado de efectuar un procedimiento de minimización formal de salida múltiple (sección 4.3.8) de las cinco funciones? No necesita construir 31 mapas de producto y pasar a través de todo el procedimiento; usted debería ser capaz de "echar un vistazo" a los mapas de excitación y salida en la sección 7.4.6 para ver qué ahorros son posibles.

máquina de memoria finita

7.56 La salida de una *máquina de memoria finita* está completamente determinada por su entrada actual y sus entradas y salidas durante los anteriores *n* tics de reloj, donde *n* es un entero finito y acotado. Cualquier máquina que puede ser realizada como se muestra en la figura X7.56 es una máquina de memoria finita. Note que una máquina de *estado* finito no necesita ser una máquina de *memoria* finita; por ejemplo, un contador módulo *n* con una entrada de habilitación y una salida "MAX" tiene solamente *n* estados, pero su salida puede depender del valor de la entrada de habilitación para cada tic de reloj desde la inicialización. Muestre cómo realizar la máquina de cerradura de combinación de la tabla 7-14 como una máquina de memoria finita.

7.57 Sintetice un circuito para el diagrama de estado ambiguo en la figura 7-62. Utilice la asignación de estado en la tabla 7-16. Escriba una lista de transición, una ecuación de transición para cada variable de estado como una suma de p términos y ecuaciones de transición/excitación simplificadas para una realización empleando flip-flops D. Determine el estado siguiente real del circuito, comenzando desde el estado IDLE, para cada una de las siguientes combinaciones de entrada en (LEFT, RIGHT HAZ): (1,0,1), (0,1,1), (1,1,0), (1,1,1). Haga comentarios sobre el comportamiento de la máquina en estos casos.

7.58 Suponga que para un estado SA y una combinación de entrada I, un ambiguo diagrama de estado indica que hay dos estados siguientes, SB y SC. El estado siguiente verdadero SD para esta transición depende de la realización de la máquina de estado. Si la máquina de estado se sintetiza utilizando el método ($V^* = \Sigma$ p términos donde $V^* = 1$) para obtener las ecuaciones de transición/excitación para flip-flops D, ¿cuál es la relación entre los estados codificados para SB, SC y SD? Explique por qué.

7.59 Repita el ejercicio 7.58, suponiendo que la máquina se sintetiza aplicando el método (V*' = Σ p términos donde V* = 0).

7.60 Supongamos que para un estado SA y una combinación de entrada I, un diagrama de estado ambiguo no define un estado siguiente. El estado siguiente verdadero SD para esta transición depende de la realización de la máquina de estado. Supongamos que la máquina de estado es sintetizada empleando el método (V* = Σ p-términos donde V* = 1) para obtener las ecuaciones de transición/excitación para flip-flops D. ¿Qué estado codificado es SD? Explique por qué.

7.61 Repita el ejercicio 7.60, suponiendo que la máquina se sintetiza aplicando el método (V*' = Σ p términos donde V* = 0).

7.62 Dadas las ecuaciones de transición para una máquina de estado sincrónica temporizada que va a construirse empleando flip-flops S-R maestro/esclavo, ¿cómo se pueden obtener las ecuaciones de excitación para las entradas S y R? (*Sugerencia:* Demuestre que cualquier ecuación de transición, Qi* = expresión, puede ser escrita en la forma Qi* = Qi · expresión 1 + Qi' · expresión 2, y vea adónde conduce esto.)

7.63 Repita el ejercicio 7.62 para flip-flops J-K. ¿Cómo se pueden especificar los elementos "sin importancia" en un diseño J-K?

7.64 Dibuje un diagrama lógico para la lógica de salida de la máquina del juego de las adivinanzas en la tabla 7-18 empleando un solo decodificador 74x139 dual de 2 a 4. (*Sugerencia:* Utilice las salidas de estado activo bajo.)

7.65 ¿Qué representa la placa personalizada de automóvil en la figura 7-60? (*Sugerencia:* Es la antigua placa del autor, una versión de ingeniero en computación de OTTFFSS.)

7.66 Analice el circuito secuencial de retroalimentación en la figura 7-19, suponiendo que las entradas PR_L y CLR_L sean siempre 1. Derive las ecuaciones de excitación, construya una tabla de transición y analice la tabla de transición para carreras críticas y no críticas. Nombre los estados, y escriba una tabla de estado/salida y una tabla de flujo/salida. Muestre que la tabla de flujo efectúa la misma función que la figura 7-85.

7.67 Dibuje el diagrama lógico para un circuito que tiene un lazo de retroalimentación pero *no* es un circuito secuencial. Es decir, la salida del circuito debería ser una función de su entrada actual solamente. Con el fin de probar su caso, divida el lazo y analice el circuito como si fuera un circuito secuencial de retroalimentación, y demuestre que las salidas para cada combinación de entrada no dependen del "estado".

flip-flop BUT
compuerta NBUT

7.68 Un *flip-flop* BUT puede ser construido a partir de una compuerta NBUT como se ilustra en la figura X7.68. (Una *compuerta* NBUT es sencillamente una compuerta BUT con salidas invertidas; véase el ejercicio 5.31 para la definición de una compuerta BUT.) Analice el flop BUT como un circuito secuencial de retroalimentación y obtenga ecuaciones de excitación, tabla de transición y tabla de flujo. ¿Este circuito es bueno para algo, o es un flop?

7.69 Repita el ejercicio 7.68 para el flop BUT en la figura X7.69.

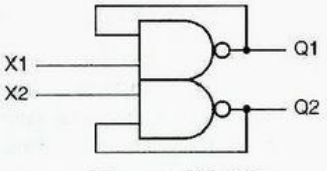


Figura X7.68

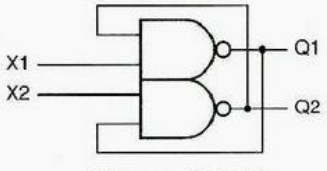


Figura X7.69

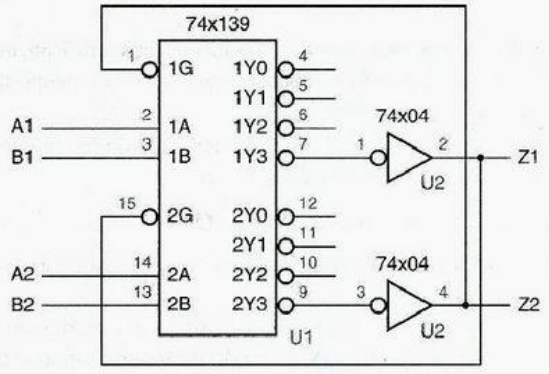


Figura X7.70

7.70 Un astuto estudiante diseñó el circuito en la figura X7.70 para crear una compuerta BUT. Pero el circuito no siempre trabaja correctamente. Analice el circuito y explique por qué.

7.71 Analice el circuito secuencial de retroalimentación de la figura X7.71. Divida los lazos de retroalimentación, escriba las ecuaciones de excitación y construya una tabla de transición y salida mostrando los estados totales estables. ¿Qué aplicación puede tener este circuito?

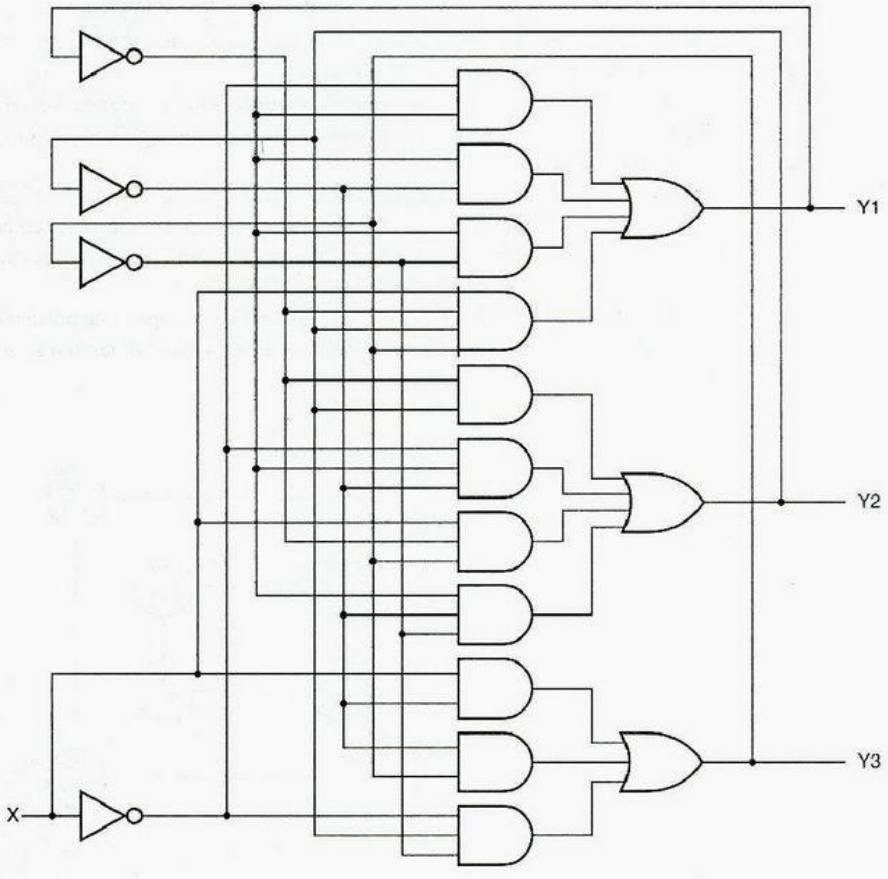


Figura X7.71

7.72 Demuestre que un sumador de complemento a 4 bit con acarreo cíclico es un circuito secuencial de retroalimentación.

7.73 Complete el análisis del flip-flop D disparado por flanco positivo en la figura 7-86, incluyendo tablas de transición/salida, estado/salida y flujo/salida. Demuestre que su comportamiento es equivalente al del flip-flop D en la figura 7-78.

7.74 Afirmamos en la sección 7.10.1 que todos los circuitos secuenciales de retroalimentación de lazo simple tienen una ecuación de excitación de la forma

$$Q^* = (\text{término de forzamiento}) + (\text{término de retención}) \cdot Q$$

¿Por qué no hay ningún circuito práctico cuya ecuación de excitación sustituya Q' en lugar de Q arriba?

7.75 Simule el circuito de cerrojo de la figura 7-88(b) bajo las condiciones descritas en el texto en la página 616, ya sea utilizando un simulador en modo de retardo unitario o bien a mano suponiendo que cada compuerta tiene un retardo de 1 ns. ¿Se comporta el circuito como se asegura en el texto? Reemplace el inversor en el circuito con tres inversores, repita la simulación y explique los resultados. ¿Qué esperaría usted que ocurriera en el circuito real?

7.76 Diseñe un latch con dos entradas de control, C1 y C2, y tres entradas de datos, D1, D2 y D3. El latch estará "abierto" solamente si ambas entradas de control son 1, y almacenará un 1 si cualquiera de las entradas de datos es 1. Haga uso de circuitos de suma de productos de dos niveles libres de riesgos para las funciones de excitación.

7.77 Repita el ejercicio 7.76, pero minimice el número de compuertas requeridas; los circuitos de excitación pueden tener múltiples niveles de lógica.

7.78 Vuelva a dibujar el diagrama de temporización en la figura 7-90, mostrando las variables de estado internas del circuito de captura de pulsos de la figura 7-100, suponiendo que comience en el estado 00.

7.79 La solución general para obtener una asignación de estado libre de carrera de 2^n estados utilizando 2^{n-1} variables de estado produce el diagrama de adyacencia mostrado en la figura X7.79 para el caso $n = 2$. Compare este diagrama con la figura 7-97. ¿Cuál es mejor, y por qué?

7.80 Diseñe una tabla de flujo en modo fundamental para un circuito de captura de pulsos similar al descrito en la sección 7.10.2, excepto que el circuito debería detectar tanto transiciones de 0 a 1 como de 1 a 0 en P.

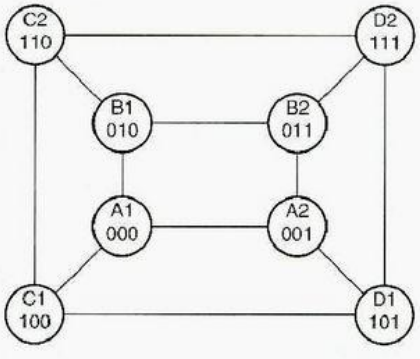


Figura X7.79

Tabla X7.85

	X Y			
S	00	01	11	10
A	B	C	—	(A)
B	(B)	E	—	(B)
C	F	(C)	—	E
D	(D)	F	—	B
E	D	(E)	—	(E)
F	(F)	(F)	—	A
	S*			

7.81 Diseñe una tabla de flujo en modo fundamental para un flip-flop D disparado por flanco positivo, uno que hace un muestreo de sus entradas y cambia sus salidas en ambos flancos de la señal del reloj.

7.82 Diseñe una tabla de flujo en modo fundamental para un circuito con dos entradas, EN y CLKIN, y una sola salida, CLKOUT, con el siguiente comportamiento. Un periodo de reloj se define como el intervalo entre flancos ascendentes sucesivos de CLKIN. Si EN es afirmada durante un periodo completo de reloj dado, entonces CLKOUT debería estar "encendido" durante el siguiente periodo de reloj; es decir, debería ser idéntico a CLKIN. Si EN es negada durante un periodo completo de reloj determinado, entonces CLKOUT debería estar "apagado" (1 constante) durante el siguiente periodo de reloj. Si EN se afirma y se niega durante un periodo de reloj dado, entonces CLKOUT debería estar encendido en el periodo siguiente si había estado apagado, y debería estar apagado si había estado encendido. Después de escribir la tabla de flujo en modo fundamental, redúzcala mediante la combinación de estados "compatibles" si es posible.

7.83 Diseñe un circuito que satisfaga las especificaciones del ejercicio 7-82 utilizando flip-flops D disparados por flanco (74x74) o flip-flops J-K (74x109) y compuertas NAND y NOR sin lazos de retroalimentación. Proporcione un diagrama de circuito completo y descripción en palabras de cómo su circuito consigue el comportamiento deseado.

7.84 ¿Cuál de los circuitos de los dos ejercicios anteriores está(n) sujeto(s) a metaestabilidad, y bajo qué condiciones?

7.85 Para la tabla de flujo en la tabla X7.85, encuentre una asignación de variables de estado que eviten todas las carreras críticas. Se pueden agregar estados adicionales como sea necesario, pero utilice tan pocas variables de estados como sea posible. Asigne la combinación de "todos 0" al estado A. Dibuje un diagrama de adyacencia para la tabla de flujo original y escriba la tabla de flujo modificada y otro diagrama de adyacencia para apoyar su asignación final de estado-variable.

7.86 Demuestre que la tabla de flujo en modo fundamental de cualquier flip-flop que muestrea entrada(s) y salidas de cambio(s) sólo en el flanco ascendente de una señal de reloj CLK contiene un riesgo esencial.

7.87 Localice el (los) riesgo(s) esencial(es) en la tabla de flujo para un flip-flop D disparado por flanco positivo, figura 7-85.

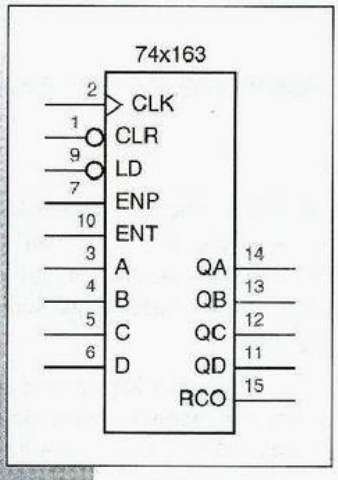
7.88 Identifique los riesgos esenciales, si existen, en la tabla de flujo desarrollada en el ejercicio 7.81.

7.89 Identifique los riesgos esenciales, si existen, en la tabla de flujo desarrollada en el ejercicio 7.82.

7.90 Construya un flip-flop verbal, un acertijo de palabras lógico que puede ser contestado correctamente en cualquiera de dos maneras dependiendo del estado. ¿Cómo podría ser adaptado un dispositivo de esta clase a la escena política?

7.91 Modifique el programa ABEL en la tabla 7-27 par utilizar una asignación de estado de salida codificada, reduciendo por ello el número total de salidas PLD requeridas por uno.

7.92 Finalice escribiendo los vectores de prueba, comenzados en la tabla 7-35, para la máquina de estado de cerradura de combinación de la tabla 7-31. El conjunto completo de vectores debería probar todas las transiciones de estado y todos los valores de salida para cada estado y combinación de entrada.



capítulo 8

Prácticas de diseño lógico secuencial

El propósito de este capítulo es familiarizarlo con los métodos de diseño de circuitos secuenciales más confiables y que se utilizan con mayor frecuencia. Por lo tanto, haremos énfasis en los *sistemas sincrónicos*, es decir, sistemas en los que todos los flip-flops (dispositivos biestables) están temporizados por la misma señal de reloj común. Aunque el mundo no marcha en su totalidad al compás de un simple reloj, dentro del ámbito de un sistema o subsistema digital, esto si es cierto. Cuando interconectamos sistemas o subsistemas digitales que utilizan relojes diferentes, por lo regular podemos identificar un número limitado de señales asincrónicas que necesitan tratamiento especial, como veremos más tarde.

Comenzaremos este capítulo con un breve resumen de estándares de documentación de circuitos secuenciales. Después de volver a visitar los bloques de construcción más básicos del diseño de circuitos secuenciales (latchs y flip-flops) describiremos algunos de los bloques de construcción más flexibles: los PLD secuenciales. Acto seguido, mostraremos cómo se realizan los contadores y registros de corrimiento tanto en dispositivos MSI como en PLD y mostraremos algunas de sus aplicaciones. Finalmente, mostraremos cómo estos elementos vienen juntos en sistemas sincrónicos y cómo se manejan las inevitables entradas asincrónicas.

8.1 Estándares de documentación de circuitos secuenciales

8.1.1 Requerimientos generales

Los estándares de documentación básicos en áreas como identificación de señal, símbolos lógicos y disposición esquemática, que introducimos en el capítulo 5, se aplican a los sistemas digitales como un todo y por lo tanto a los circuitos secuenciales en particular. Sin embargo, destacaremos las siguientes ideas para elementos del sistema que son específicamente "secuenciales":

- *Formato de la máquina de estado.* Dentro de un diagrama lógico, una colección de flip-flops y lógica combinacional que forman una máquina de estado deberían dibujarse juntos en un formato lógico en la misma página, de manera que el hecho de que sea una máquina de estado sea obvio (¡Usted no debería pasar más páginas para encontrar la trayectoria de retroalimentación!)
- *Elementos en cascada.* De manera similar, los registros, contadores y registros de corrimiento que utilizan múltiples CI deberán tener estos CI agrupados en el esquema de modo que la estructura en cascada sea obvia.
- *Flip-flops (dispositivos biestables).* Los símbolos para los elementos individuales del circuito secuencial, en especial los flip-flops, deberán seguir rigurosamente los estándares de dibujo apropiados, de manera que el tipo, la función y el comportamiento temporal de los elementos sean claros.
- *Descripciones de la máquina de estado.* Las máquinas de estado deberán describirse mediante tablas de estado, diagramas de estado, listas de transición o archivos de texto en un lenguaje de descripción de máquina de estado como ABEL o VHDL.
- *Diagramas de temporización.* El paquete de documentación para circuitos secuenciales debería incluir diagramas de temporización que muestren las suposiciones generales de tiempo y el comportamiento temporal del circuito.
- *Especificaciones de temporización.* Un circuito secuencial deberá estar acompañado de una especificación de los requerimientos de tiempo para una adecuada operación interna (por ejemplo, máxima frecuencia de reloj). Además de los requerimientos para cualquier entrada suministrada externamente (por ejemplo, requerimientos de tiempo de establecimiento y tiempo de retención con respecto al reloj del sistema, anchuras de pulso mínimas, etcétera).

8.1.2 Símbolos lógicos

Presentamos los símbolos tradicionales para flip-flops en la sección 7.2. Los flip-flops siempre se dibujan como símbolos de forma rectangular y siguen las mismas pautas generales que otros símbolos de forma rectangular: entradas a la izquierda, salidas a la derecha, burbujas para niveles activos, y así sucesivamente. Además, algunas pautas específicas se aplican a los símbolos de los flip-flops:

- Un indicador dinámico se coloca en las entradas de reloj disparadas por flanco.
- Un indicador de salida pospuesta se coloca en las salidas maestro/esclavo que cambian en el intervalo final durante el cual el reloj es asertivo.
- Las entradas de preestablecimiento asincrónico y de borrado pueden mostrarse en la parte superior e inferior de un símbolo de flip-flop, prestablecimiento en la parte superior y borrado en la parte inferior.

Los símbolos lógicos para elementos secuenciales de mayor escala, como los contadores y registros de corrimiento descritos posteriormente en este capítulo, generalmente se dibujan con todas las entradas, incluyendo prestablecimientos y borrados, a la izquierda, y todas las salidas a la derecha. Las señales bidireccionales pueden dibujarse a la izquierda o a la derecha, donde quiera que sea conveniente.

Como los flip-flops individuales, los elementos secuenciales de mayor escala utilizan un indicador dinámico para señalar entradas de reloj disparadas por flanco. En los símbolos "tradicionales", los nombres de las entradas y salidas daban una pista de sus funciones, pero son a veces ambiguos. Por ejemplo, dos elementos que se describen más adelante en este capítulo, los contadores de 4 bits 74x161 y 74x163, tienen exactamente el mismo símbolo tradicional, aun cuando el comportamiento de sus entradas CLR sea completamente diferente.

SÍMBOLOS ESTÁNDAR IEEE Los símbolos estándar IEEE tienen un rico conjunto de notación que puede suministrar una definición no ambigua de cada función de señal. Una guía para los símbolos IEEE, incluyendo los de los elementos secuenciales en este capítulo, pueden encontrarse en la red de internet www.ddpp.com.

8.1.3 Descripciones de máquina de estado

Hasta ahora hemos tratado con seis diferentes representaciones de máquinas de estado:

- Descripción
- Tablas de estado
- Diagramas de estado
- Listas de transición
- Programas ABEL
- Programas VHDL

Usted puede pensar que tener todas estas diferentes formas de representar máquinas de estado es un problema: ¡demasiadas cosas por aprender! Bien, no todas ellas son difíciles de aprender, pero *hay* aquí un problema sutil.

Considere un problema similar en programación, donde el "pseudocódigo" de alto nivel o quizás un diagrama de flujo puede utilizarse para documentar cómo funciona un programa. El pseudocódigo puede expresar las intenciones del programador muy bien, pero pueden presentarse errores, malinterpretaciones y fallas tipográficas cuando se traduce al código real. En cualquier proceso creativo, pueden presentarse inconsistencias cuando existen múltiples representaciones de cómo funcionan las cosas.

representaciones inconsistentes de máquina de estado

La misma clase de inconsistencias puede ocurrir en el diseño de máquina de estado. Un diseñador lógico puede documentar el comportamiento deseado de una máquina con un diagrama de estado hecho a mano 100% correcto, pero puede cometer equivocaciones al trasladar el diagrama a un programa, y existen *muchas* oportunidades para estropearlo si usted tiene que realizar cálculos manualmente para traducir el diagrama de estado a una tabla de estado, tabla de transición, ecuaciones de excitación y diagrama lógico.

La solución a este problema es semejante a la adoptada por los programadores que escriben código autodocumentado utilizando un lenguaje de alto nivel. La clave es seleccionar una representación que exprese las intenciones del diseñador *y que además* sea traducible en una realización física empleando un proceso automatizado, libre de errores. (Usted no ha escuchado a muchos programadores gritar "¡basura de compilador!" cuando sus programas no funcionan a la primera vez.)

La mejor solución (por ahora, al menos) es escribir "programas" de máquina de estado directamente en un lenguaje de descripción de máquina de estado de alto nivel como ABEL o VHDL y evitar representaciones alternas, aparte de las descripciones en palabras de resumen general. Los lenguajes como ABEL y VHDL son fácilmente legibles y permiten la conversión automática de la descripción en una realización basada en PLD, FPGA o ASIC. Algunas herramientas de CAD permiten la especificación y síntesis de las máquinas de estado mediante el uso de diagramas de estado o incluso mediante diagramas de temporización de muestra, pero a menudo éstos pueden generar ambigüedades y resultados no deseados. De este modo, haremos uso del enfoque ABEL/VHDL exclusivamente en el resto de este libro.

8.1.4 Especificaciones y diagramas de tiempo

Mostramos muchos ejemplos de diagramas de temporización en los capítulos 5 y 7. En el diseño de sistemas asincrónicos, la mayoría de los diagramas de temporización muestran la relación entre el reloj y diversas señales de entrada, salida e internas.

La figura 8-1 muestra un diagrama de temporización bastante típico que especifica los requerimientos y características de señales de entrada y salida en un circuito sincrónico. La primera línea muestra el reloj del sistema y sus parámetros de tiempo nominales. Las líneas restantes muestran un intervalo de retardos para otras señales.

Por ejemplo, la segunda línea muestra que los flip-flops cambian sus salidas en algún momento entre el flanco ascendente de CLOCK y el tiempo t_{ffpd} después. Los circuitos externos que muestrean estas señales no deberán hacerlo mientras estén cambiando. El diagrama de temporización se dibuja como si el valor mínimo de t_{ffpd} fuera cero; un

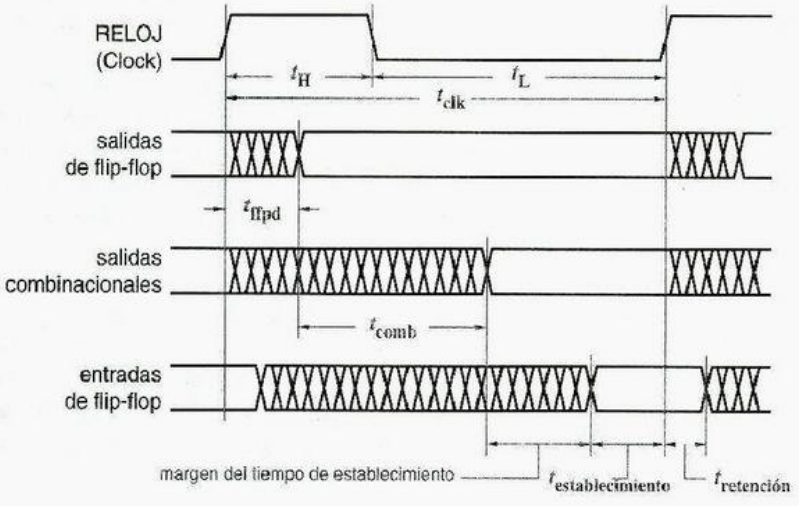


Figura 8-1
Un diagrama de temporización detallado que muestra los retardos de propagación y los tiempos de establecimiento y de retención con respecto al reloj.

paquete de documentación completo incluiría una tabla de temporización indicando los valores mínimo, típico y máximo reales de t_{ffpd} y otros parámetros de temporización.

La tercera línea del diagrama de temporización muestra el tiempo adicional, t_{comb}, requerido para que los cambios de salida de flip-flop se propaguen a través de los elementos lógicos combinacionales, tal como la lógica de excitación del flip-flop. Las entradas de excitación de los flip-flops y otros dispositivos temporizados requieren un tiempo de establecimiento de $t_{establecimiento}$, como se muestra en la cuarta línea. Para una adecuada operación del circuito debemos tener $t_{clk} - t_{ffpd} - t_{comb} > t_{establecimiento}$.

margen de temporización

Los márgenes de temporización indican en qué medida los componentes individuales de un circuito pueden ser "peores que en el peor de los casos" sin provocar que el circuito falle. Los sistemas bien diseñados tienen márgenes de temporización positivos, distintos de cero, para tener en cuenta circunstancias inesperadas (componentes marginales, apagones parciales, errores de ingeniería, etc.) y sesgo de reloj (sección 8.8.1).

margen de tiempo de establecimiento

El valor $t_{clk} - t_{ffpd(max)} - t_{comb(max)} - t_{establecimiento}$ se denomina el *margen de tiempo de establecimiento*; si éste es negativo, el circuito no funcionará. Advierta que se utilizan retardos de propagación *máximos* para calcular el margen de tiempo de establecimiento. Otro margen de temporización involucra el requerimiento de tiempo de retención, $t_{retención}$; la suma de los valores *mínimos* de t_{ffpd} y t_{comb} debe ser mayor que $t_{retención}$, y el *margen de tiempo de retención* es $t_{ffpd(min)} + t_{comb(min)} - t_{retención}$.

margen de tiempo de retención

El diagrama de temporización en la figura 8-1 no muestra las diferencias de tiempo entre diferentes entradas de flip-flops o señales de lógica combinacional, aun cuando tales diferencias existen en la mayoría de los circuitos. Por ejemplo, una salida Q del flip-flop puede estar conectada directamente a la entrada D de otro flip-flop, de modo que t_{comb} para esa trayectoria es cero, mientras que otra puede ir en la trayectoria de propagación de acarreo un sumador de 32 bits antes de alcanzar una entrada de flip-flop. Cuando se utiliza la metodología adecuada de diseño sincrónico, estos tiempos relativos no son críticos, puesto que ninguna de estas señales afecta el estado del circuito hasta que se presenta un flanco de reloj. Usted simplemente tiene que hallar la trayectoria de retardo más larga en un periodo de reloj para determinar si el circuito funcionará. Sin embargo, puede que tenga que analizar varias trayectorias diferentes a fin de encontrar el peor de los casos.

Otro tipo de diagrama de temporización más común muestra solamente el comportamiento funcional y no está interesado en cantidades de retardo reales; un ejemplo se muestra en la figura 8-2. Aquí, el reloj es "perfecto". El hecho de mostrar cambios de señal como líneas verticales o inclinadas es estrictamente cuestión de gustos personales en este y otros diagramas de temporización, a menos que los tiempos de ascenso y caída deban ser indicados explícitamente. Las transiciones de reloj se muestran como líneas verticales en ésta y otras figuras manteniendo la idea de que el reloj es una señal de referencia "perfecta".

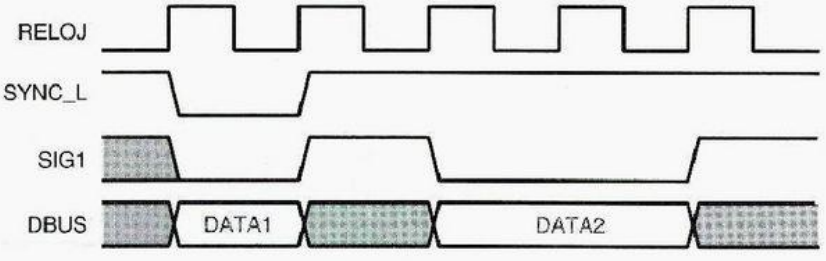


Figura 8-2
Temporización funcional de un circuito sincrónico.

NADA ES PERFECTO

En realidad, no hay tal cosa como una señal de reloj perfecta. Una imperfección que la mayoría de los diseñadores de circuitos digitales de alta velocidad tiene que enfrentar es el "sesgo de reloj". Como lo mostramos en la sección 8.8.1, un flanco de reloj dado llega a diferentes entradas de circuito en diferentes tiempos debido a las diferencias en los retardos del alambrado, carga y otros efectos.

Otra imperfección, un poco más allá del alcance de este texto, es la "fluctuación (jitter) de reloj". Un reloj de 10 MHz no tiene un periodo de exactamente 100 ns en cada ciclo: puede ser de 100.05 ns en un ciclo y de 99.5 ns en el siguiente. Esto no es nada del otro mundo en lo que concierne a un circuito lento, pero en un circuito de 500 MHz el mismo 0.1 ns de fluctuacion consume hasta 5% de los 2 ns disponibles por ciclo. ¡Y la fluctuación en algunas fuentes de reloj es aún más alta!

Las otras señales en la figura 8-2 pueden ser las salidas de flip-flops, salidas combinacionales o entradas de flip-flops. El sombreado se utiliza para indicar valores de señal "sin importancia"; sombrear como en la figura 8-1 podría usarse en su lugar. Todas las señales se muestran cambiando inmediatamente después del flanco de reloj. En realidad, las salidas cambian un poco después, y las entradas pueden cambiar justo apenas antes del siguiente flanco de reloj. Sin embargo, "alineando" todo sobre el flanco del reloj permite al diagrama de temporización exhibir con más claridad cuáles funciones se efectúan durante cada periodo de reloj. Las señales que están alineadas con el reloj son interpretadas como que cambian en algún momento *después* del flanco del reloj, con tiempo que satisface los requerimientos del tiempo de establecimiento y retención del circuito. Muchos diagramas de temporización de este tipo aparecen en este capítulo.

La tabla 8-1 muestra los parámetros de tiempo del fabricante de flip-flops, registros y cerrojos. Estos parámetros se utilizan con bastante frecuencia en los dispositivos

Tabla 8-1 Retardo de propagación en ns correspondiente a latchs, registros y flip-flops CMOS seleccionados.

		74HCT		*74AHCT*		*74FCT*		*74LS*	
Parte	***Parámetro***	***Tipo***	***Max.***	***Tipo***	***Máx.***	***Mín.***	***Máx.***	***Tipo***	***Máx.***
'74	t_{pd}, CLK↑ a Q o $\overline{Q}$	35	44	6.3	10			25	40
	t_{pd}, $\overline{PR}$↓ o $\overline{CLR}$↓ a Q o $\overline{Q}$	40	50	8.1	13			25	40
	t_s, D a CLK↑	12	15		5				20
	t_h, D de CLK↑	3	3		0				5
	t_{rec}, CLK↑ de $\overline{PR}$↑ o $\overline{CLR}$↑	6	8		3.5				
	t_w, CLK bajo o alto	18	23		5				25
	t_w, $\overline{PR}$ o $\overline{CLR}$ bajo	16	20		5				25
'174	t_{pd}, CLK↑ a Q	40	50	6.3	10			21	30
	t_{pd}, $\overline{CLR}$↓ a Q	44	55	8.1	13			23	35
	t_s, D a CLK↑	16	20		5				20
	t_h, D de CLK↑	5	5		0				5
	t_{rec}, CLK↑ de $\overline{CLR}$↑	12	15		3.5				25
	t_w, CLK bajo o alto	20	25		5				20
	t_w, $\overline{CLR}$ bajo	25	31		5				20

Tabla 8-1 (continuación) Retardo de propagación en ns correspondiente a latchs, registros y flip-flops CMOS seleccionados.

		74HCT		74AHCT		74FCT		74LS	
Parte	**Parámetro**	**Tipo**	**Máx.**	**Tipo**	**Máx.**	**Mín.**	**Máx.**	**Tipo**	**Máx.**
'175	t_{pd}, CLK↑ a Q o $\overline{Q}$	33	41					21	30
	t_{pd}, $\overline{CLR}$↓ a Q o $\overline{Q}$	35	44					23	35
	t_s, D a CLK↑	20	25						20
	t_h, D de CLK↑	5	5						5
	t_{rec}, CLK↑ de $\overline{CLR}$↑	5	5						25
	t_w, CLK bajo o alto	20	25						20
	t_w, $\overline{CLR}$ bajo	20	25						20
'273	t_{pd}, CLK↑ a Q	30	38	6.8	11	2	7.2	18	27
	t_{pd}, $\overline{CLR}$↓ a Q	32	40	8.5	12.6	2	7.2	18	27
	t_s, D a CLK↑	12	15		5		2		20
	t_h, D de CLK↑	3	3		0		1.5		5
	t_{rec}, CLK↑ de $\overline{CLR}$↑	10	13		2.5		2		25
	t_w, CLK bajo o alto	20	25		6.5		4		20
	t_w, $\overline{CLR}$ bajo	12	15		6		5		20
'373	t_{pd}, C↑ a Q	35	44	8.5	14.5	2	8.5	24	36
	t_{pd}, D a Q	32	40	5.9	10.5	1.5	5.2	18	27
	t_s, D a C↓	10	13		1.5		2		0
	t_h, D de C↓	5	5		3.5		1.5		10
	t_{pHZ}, $\overline{OE}$ a Q	35	44		12	1.5	6.5	12	20
	t_{pLZ}, $\overline{OE}$ a Q	35	44		12	1.5	6.5	16	25
	t_{pZH}, $\overline{OE}$ a Q	35	44		13.5	1.5	5.5	16	28
	t_{pZL}, $\overline{OE}$ a Q	35	44		13.5	1.5	5.5	22	36
	t_w, C bajo o alto	16	20		6.5		5		15
'374	t_{pd}, CLK↑ a Q	33	41	6.4	11.5	2	6.5	22	34
	t_s, D a CLK↑	12	15		2.5		2		20
	t_h, D de CLK↑	5	5		2.5		1.5		0
	t_{pHZ}, $\overline{OE}$ a Q	28	35		12	1.5	6.5		18
	t_{pLZ}, $\overline{OE}$ a Q	28	35		12	1.5	6.5		24
	t_{pZH}, $\overline{OE}$ a Q	30	38		12.5	1.5	5.5		28
	t_{pZL}, $\overline{OE}$ a Q	30	38		12.5	1.5	5.5		36
	t_w, CLK bajo o alto	16	20		6.5		5		15
'377	t_{pd}, CLK↑ a Q	38	48			2	7.2	18	27
	t_s, D a CLK↑	12	15				2		20
	t_h, D de CLK↑	3	3				1.5		5
	t_s, $\overline{EN}$ a CLK↑	12	15				2		25
	t_h, $\overline{EN}$ de CLK↑	5	5				1.5		5
	t_w, CLK bajo o alto	20	25				6		20

CMOS y TTL. Estos son los valores "típicos" para los dispositivos que operan a 25°C, pero, dependiendo de la familia lógica, podrían ser para una parte típica y voltaje de alimentación nominal, o podrían ser para una parte en el peor de los casos, para el voltaje de alimentación más desfavorable. Los valores "máximos" son generalmente en todo el intervalo de operación comercial de voltaje y temperatura, excepto valores TTL, los cuales se encuentran especificados a 25°C. También note que los valores "máximos" de t_s, t_h, t_{rec} o t_w son los valores máximos del *mínimo* tiempo de establecimiento, tiempo de retención, tiempo de recuperación o anchura de pulso que la parte especificada exhibirá.

Distintos fabricantes pueden emplear definiciones ligeramente diferentes para los mismos parámetros de tiempo, y pueden especificar diferentes valores para la misma parte. Un fabricante determinado puede incluso emplear diferentes definiciones para diferentes familias o números de parte en la misma familia. De este modo, todas las especificaciones en la tabla 8-1 son meramente representativas; para números exactos y sus definiciones, usted debe consultar la hoja de datos para la parte y el fabricante en particular.

8.2 Latchs y flip-flops

8.2.1 Latchs y flip-flops SSI

Diversos tipos de latchs y flip-flops discretos se encuentran disponibles como partes tipo SSI. Estos dispositivos en ocasiones son empleados en el diseño de máquinas de estado y circuitos secuenciales "no estructurados" que no caen dentro de las categorías de registros de acarreo, contadores y otras funciones MSI secuenciales presentadas posteriormente en este capítulo. Sin embargo, los latchs y flip-flops SSI han sido eliminados en gran medida en los diseños modernos a medida que sus funciones se integran en los PLD y los FPGA. No obstante, un puñado de estos bloques de construcción discretos aún aparece en muchos sistemas digitales, de modo que es importante estar familiarizados con ellos.

74x375

La figura 8-3 muestra las terminales de salida para diversos dispositivos secuenciales SSI. El único latch en la figura es el *74x375*, que contiene cuatro latchs D, similares en su función a los latchs D "genéricos" descritos en la sección 7.2.4. Debido a las limitaciones de terminal, los latchs se encuentran acomodados en pares con una línea de control C común para cada par.

74x74

Entre los dispositivos en la figura 8-3, el más importante es el *74x74*, que contiene dos flip-flops independientes D disparados por flanco positivo con entradas de preestablecimiento y de borrado. Describimos la operación funcional, temporización y estructura interna de los flip-flops D disparados por flanco positivo en general, y del 74x74 en particular, en la sección 7.2.5. Aparte del uso del 74x74 en circuitos secuenciales "variados", las versiones rápidas de la parte, tales como *74F74* y *74ACT74*, encuentran aplicaciones en sincronizadores para señales de entrada asincrónicas, como se discute en la sección 8.9.

74F74
74ACT74

74x109

El *74x109* es un flip-flop J-$\overline{K}$ disparado por flanco positivo con una entrada K de nivel activo bajo (nombrada $\overline{K}$ o K_L). Discutimos la estructura interna del '109 en la sección 7.2.10. Otro flip-flop J-K es el *74x112*, el cual tiene una entrada de reloj de nivel activo bajo.

74x112

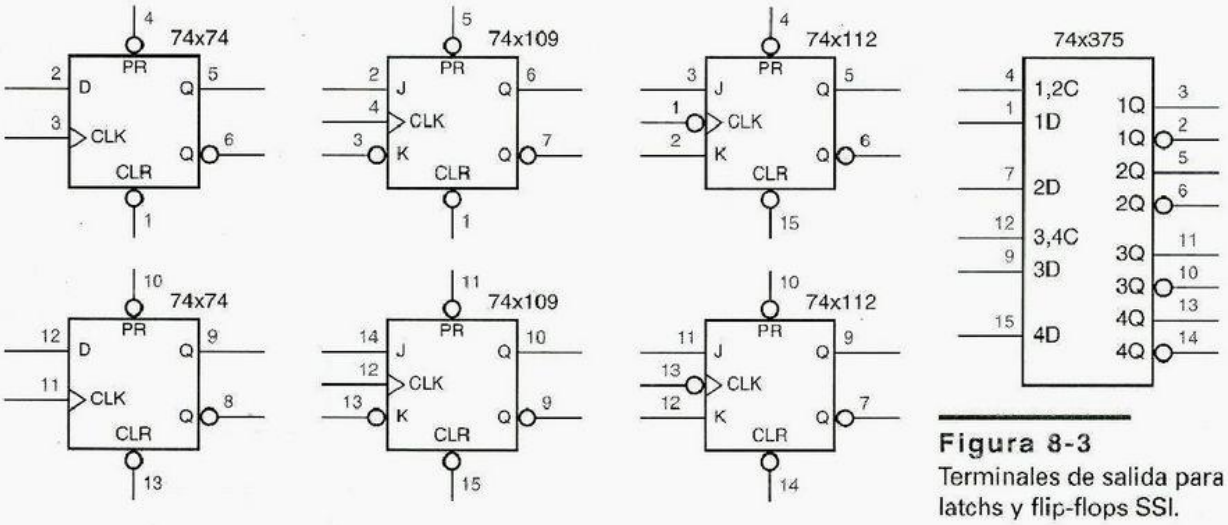


Figura 8-3
Terminales de salida para latchs y flip-flops SSI.

*8.2.2 Inhibición del rebote de un interruptor

Una aplicación común de los flip-flops y latchs simples es la inhibición de rebote de un interruptor. Todos estamos familiarizados con los interruptores eléctricos gracias a nuestra experiencia con lámparas, trituradores de basura y otros aparatos. Los interruptores conectados a fuentes de lógica constante 0 y 1 se utilizan con frecuencia en sistemas digitales para suministrar "entradas del usuario". Sin embargo, en aplicaciones lógicas digitales debemos considerar otro aspecto de la operación del interruptor, la dimensión del tiempo. Una simple operación de encendido-apagado que ocurre instantáneamente hasta donde concierne a nosotros humanos de lento movimiento, en realidad tiene varias fases que son diferenciadas por la lógica digital de alta velocidad.

La figura 8-4(a) muestra cómo un interruptor monopolar de una vía (SPST, *single-pole, single-throw*) puede utilizarse para generar una entrada lógica simple. Un resistor de arranque proporciona un valor lógico 1 cuando el interruptor se abre, y el contacto del interruptor se une a tierra para proporcionar un valor lógico 0 cuando el interruptor es oprimido.

Como se muestra en (b), se necesita un momento, después de activar la palanca de conmutación, para que ésta toque al contacto inferior. Una vez que lo hace, no permanece así mucho tiempo; rebota varias veces antes de asentarse. El resultado es que se observan varias transiciones en las señales lógicas SW_L y DSW para cada activación del interruptor. Este comportamiento se conoce como *rebote de contacto*. Los interruptores típicos rebotan durante 10-20 ms, un tiempo muy largo en comparación con las velocidades de conmutación de las compuertas lógicas.

rebote de contacto

El rebote de contacto puede o no ser un problema, dependiendo de la aplicación del interruptor. Por ejemplo, algunas computadoras tienen información de configuración especificada por pequeños interruptores, denominados *interruptores tipo DIP* porque son del mismo tamaño de un encapsulado en línea doble (DIP, por sus siglas en inglés). Puesto que los interruptores DIP son normalmente cambiados solamente cuando la computadora se encuentra inactiva, no hay problema. El rebote de contacto *es* un problema si

interruptor DIP

* En este libro, las secciones opcionales están marcadas con un asterisco.

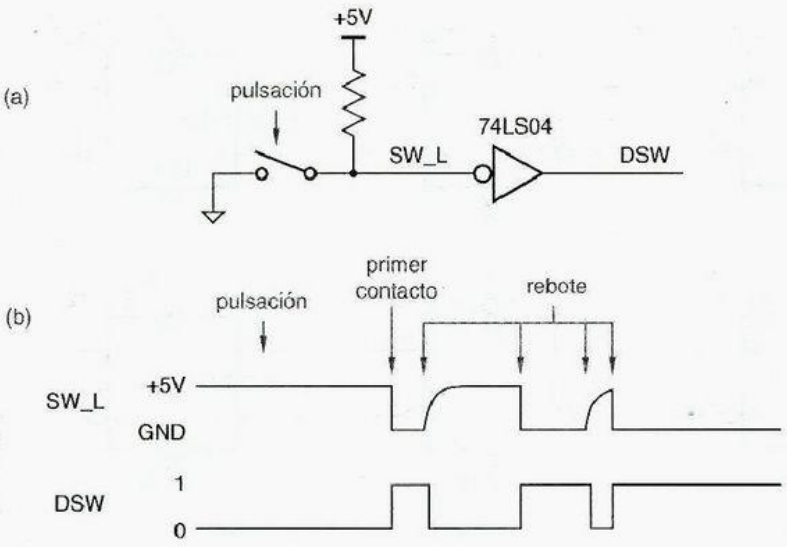


Figura 8-4
Entrada de interruptor sin rebote.

un interruptor se utiliza para contar o señalizar un evento (por ejemplo, las vueltas en una carrera). Entonces debemos proporcionar un circuito (o, en los sistemas basados en microprocesadores, software) para *inhibir el rebote* del interruptor: proporcionar precisamente una cambio de señal o pulso para cada evento externo.

inhibición del rebote

*8.2.3 El inhibidor de rebote para el interruptor más simple

La inhibición del rebote del interruptor es una buena aplicación para el circuito secuencial más simple, el elemento biestable de la sección 7.1, el cual puede utilizarse como se muestra en la figura 8-5. Este circuito utiliza un interruptor monopolar, de doble vía (SPDT, single-pole, double-switch). Los contactos del interruptor y el brazo conmutador tienen un comportamiento de "romper antes de hacer", de modo que las terminales del brazo están "flotando" durante un tiempo, a la mitad de la trayectoria de activación del interruptor.

Antes de que se presione el botón, el contacto superior mantiene SW a 0 V, un 0 lógico válido, y el inversor superior produce un 1 lógico en SW_L y en el contacto inferior. Cuando presionamos el botón y se rompe el contacto, la retroalimentación en el flip-flop mantiene SW a V_{OL} (≤ 0.5 V para LS-TTL), que todavía es un 0 lógico válido.

Acto seguido, cuando el brazo toca el contacto inferior, el circuito funciona de manera bastante poco convencional por un momento. El inversor superior en flip-flop está intentando mantener un 1 lógico en la señal SW_L; el transistor superior en su salida de totem-polo se encuentra "encendida" y conectando SW_L a través de una pequeña resistencia a +5 V. Súbitamente, el contacto del interruptor efectúa una conexión metálica de SW_L a tierra, 0.0 V. Como es lógico, gana el contacto del interruptor.

Poco tiempo después (30 ns para el 74LS04), el 0 lógico forzado en SW_L se propaga a través de los dos inversores del flip-flop, de modo que el inversor superior abandona su vano intento de conducir un 1, y en su lugar conduce un 0 lógico sobre SW_L. En este punto, la salida del inversor superior ya no se pone en corto a tierra, y la retroalimentación en el flip-flop mantiene el 0 lógico en SW_L incluso si el brazo rebota fuera del contacto inferior, como lo hace. (No rebota lo suficiente para tocar el contacto superior de nuevo.)

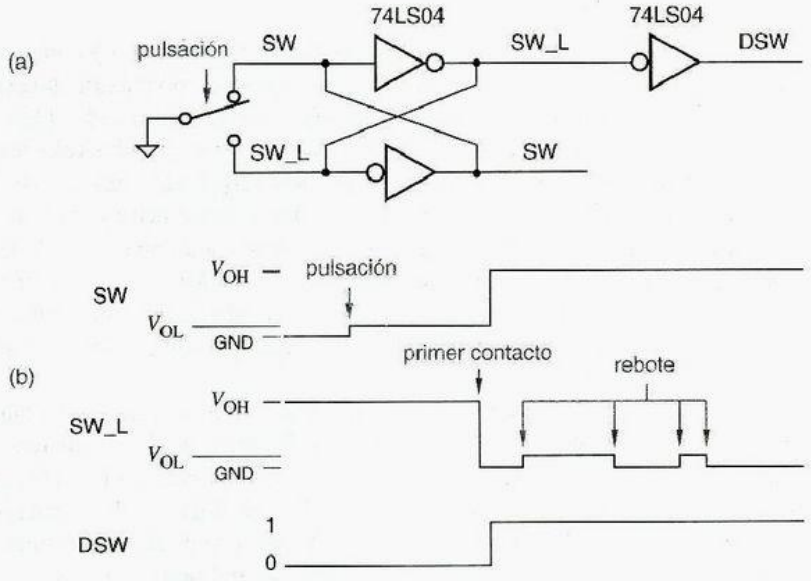


Figura 8-5
Entrada de interruptor utilizando un flip-flop para la inhibición del rebote.

Las ventajas de este circuito en comparación con otros enfoques de inhibición de rebote son que tiene un conteo de chips más bajo (una tercera parte de un 74LS04), no se requieren resistores de arranque y se producen ambas polaridades (nivel activo alto y nivel activo bajo) de la señal de entrada. En situaciones donde momentáneamente debe evitarse el corto de las salidas de compuerta, puede diseñarse un circuito semejante que utiliza un latch $\overline{S}$-$\overline{R}$ y resistores de arranque, como se sugiere en la figura 8-6.

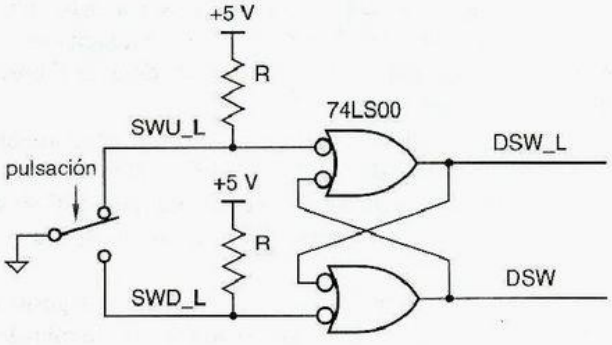


Figura 8-6
Entrada del interruptor empleando un latch $\overline{S}$-$\overline{R}$ para la inhibición del rebote.

DONDE LO DÉBIL TRABAJA BIEN

El circuito en la figura 8-5, al mismo tiempo que es elegante, no debería utilizarse con dispositivos CMOS de alta velocidad, como el 74ACT04, cuyas salidas son capaces de suministrar grandes cantidades de corriente en el estado ALTO (HIGH). Mientras que poner en corto tales salidas a tierra momentáneamente no provocará daño alguno, en cambio generará un pulso de ruido en las señales de alimentación y tierra que puede disparar una operación impropia del circuito en cualquier parte. El circuito de inhibición de rebote en la figura funciona bien con familias lógicas más débiles como HCT y LS-TTL.

*8.2.4 Circuito retenedor de bus

En las secciones 3.7.3 y 5.6 describimos las salidas de tres estados y cómo se conjuntan para crear buses de tres estados. En cualquier momento, como máximo una salida puede controlar el bus; en ocasiones, ninguna salida se encuentra controlando el bus, y el bus se encuentra "flotando". Cuando se conectan entradas CMOS de alta velocidad a un bus que se deja flotando por largo tiempo (en los circuitos más rápidos, más de uno o dos tics de reloj), pueden ocurrir cosas malas. En particular, ruido, interferencias y otros efectos pueden conducir las señales del bus flotante de alta impedancia a un nivel de voltaje cercano al umbral de conmutación de entrada de los dispositivos CMOS, lo que a su vez permite un excesivo flujo de corriente en las salidas del dispositivo. Por esta razón, es deseable y acostumbrado proporcionar resistores de arranque que rápidamente arrastren un bus flotante a un nivel lógico ALTO válido.

Los resistores de arranque no son toda bondad: cuestan dinero y ocupan valioso espacio real en las placas de circuitos impresos. Un gran problema que tenemos en circuitos de muy alta velocidad es la elección del valor de la resistencia. Si la resistencia es demasiado alta, cuando un bus va del nivel BAJO al de flotación, la transición de BAJO al nivel de arranque ALTO será lenta debido a la constante de tiempo RC alta, y los niveles de entrada pueden pasar demasiado tiempo cerca del umbral de conmutación. Si la resistencia de arranque es demasiado baja, los dispositivos que intenten arrastrar el bus en nivel BAJO tendrán que consumir demasiada corriente.

circuito de retención de bus

La solución a este problema es eliminar los resistores de arranque a favor de un *circuito retenedor de bus* activo, ilustrado en la figura 8-7. Esto no es más que un flip-flop con un resistor R en una rama del lazo de retroalimentación. La señal INOUT del retenedor de bus se conecta a la línea de bus de tres estados que se retendrá. Cuando la salida de tres estados que actualmente controla la línea BAJO o ALTO cambia a flotación, el inversor derecho del retenedor de bus mantiene la línea en su estado actual. Cuando una salida de tres estados intenta cambiar la línea de BAJO a ALTO o viceversa, debe suministrar o consumir una pequeña cantidad de corriente adicional a través de R para superar el retenedor de bus. Este flujo de corriente adicional persiste solamente durante el breve tiempo que le toma al flip-flop pasar a su otro estado estable.

La elección del valor de R en el retenedor de bus es un compromiso entre la corriente de cancelación baja (R alta) y una buena inmunidad al ruido en la línea del retenedor de bus (R baja). Un ejemplo típico, los circuitos de retención de bus en la familia LVC CMOS de 3.3 V especifican una corriente de cancelación máxima de 500 μA, lo que implica que $R \approx 3.3 / 0.0005 = 6.6\ \text{K}\Omega$.

Los circuitos de retención de bus se fabrican con frecuencia dentro de otro dispositivo MSI, tal como un controlador o transceptor de bus CMOS octal. No requieren ni terminales ni área adicional en el chip, de manera que son esencialmente gratuitos. No hay problema si se tienen varios retenedores de bus (n) en la misma línea de señal, siempre que los retenedores de bus sean capaces de proporcionar n veces la corriente de cancelación durante el tiempo que dura la conmutación. Note que los retenedores de bus normalmente no son efectivos en buses que tienen entradas TTL vinculadas a ellos (vea el ejercicio 8.25).

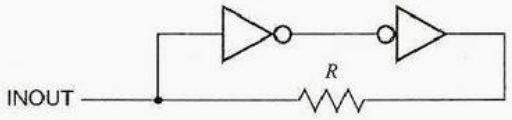


Figura 8-7
Circuito de retención de bus.

8.2.5 Registros y latchs de bits múltiples

Una colección de dos o más flip-flops D con una entrada de reloj en común se denomina un *registro*. Los registros se utilizan con frecuencia para almacenar una colección de bits relacionados, tal como un byte de datos en una computadora. Sin embargo, un registro simple también puede ser empleado para almacenar bits de datos sin relación o información de control; la única restricción real es que todos los bits se almacenan utilizando la misma señal de reloj.

registro

La figura 8-8 muestra el diagrama lógico y el símbolo lógico para un registro MSI que se utiliza con frecuencia, el *74x175*. El 74x175 contiene cuatro flip-flops D disparados por flanco positivo con un reloj común y entradas de borrado asincrónicas. Proporciona tanto salidas de nivel activo alto como de nivel activo bajo en las terminales externas del dispositivo.

74x175

Los flip-flops individuales en un '175 son disparados por flanco negativo, como lo indican las burbujas de inversión en sus entradas CLK. Sin embargo, el circuito también contiene un inversor que hace los flip-flops disparados por flanco positivo con respecto a la terminal de entrada CLK externa del dispositivo. La señal de borrado común, de nivel activo bajo (CLR_L) está conectada a las entradas de borrado asincrónicas de todos los cuatro flip-flops. Tanto CLK como CLR_L pasan por buffers antes de distribuirse a los cuatro flip-flops, de modo que un dispositivo que controle una de estas entradas ve solamente una carga unitaria en vez de cuatro. Esto es especialmente importante si una señal de borrado o reloj común debe controlar muchos registros así.

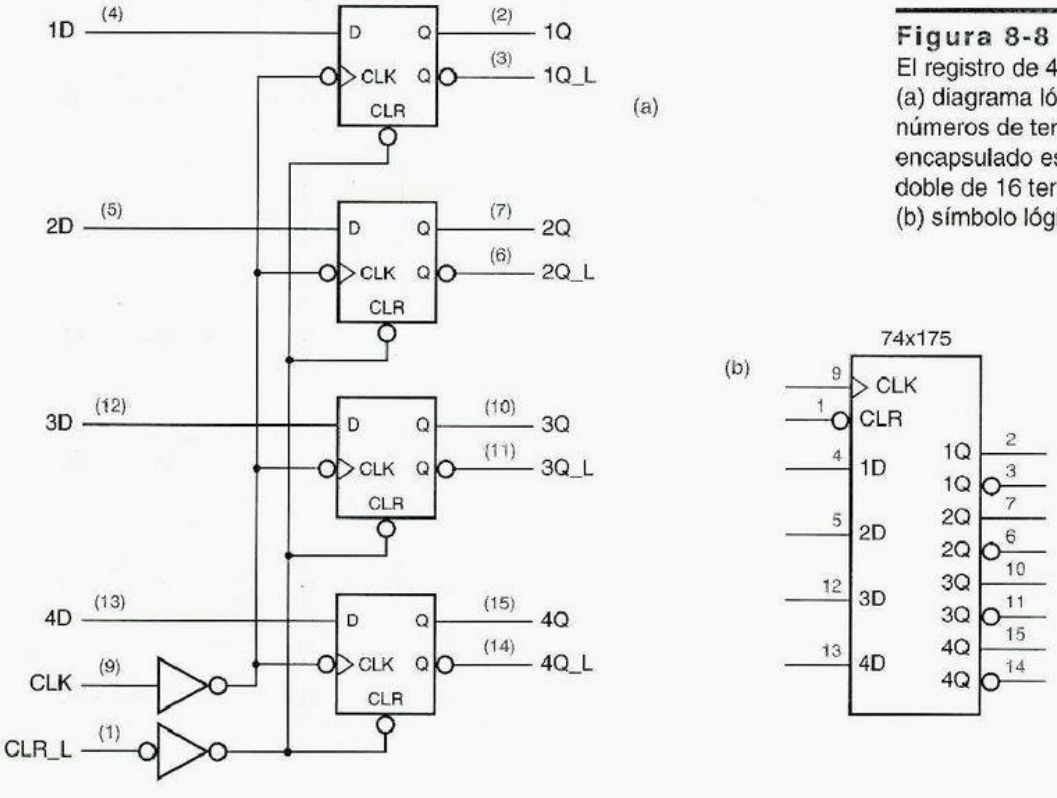


Figura 8-8
El registro de 4 bits 74x175: (a) diagrama lógico, incluyendo números de terminales para un encapsulado estándar en línea doble de 16 terminales; (b) símbolo lógico tradicional.

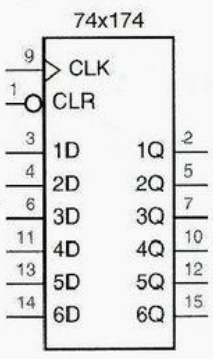


Figura 8-9
Símbolo lógico para el registro de 6 bits 74x174.

El símbolo lógico para el registro de 6 bits, *74x174*, se ilustra en la figura 8-9. La estructura interna de este dispositivo es semejante a la del 74x175, excepto que elimina las salidas de nivel activo bajo y proporciona en su lugar dos flip-flops más.

Muchos sistemas digitales, incluyendo computadoras, dispositivos de telecomunicación y equipos estereofónicos, procesan información de 8, 16 o 32 bits a la vez; como resultado, son muy populares los CI que manejan 8 bits. Un CI MSI de esta clase es el flip-flop D octal disparado por flanco *74x374*, también conocido simplemente como un registro de 8 bits. (Una vez más, "octal" significa que el dispositivo tiene ocho secciones.)

Como se muestra en la figura 8-10, el 74x374 contiene ocho flip-flops D disparados por flanco que muestrean sus entradas y cambian sus salidas en el flanco ascendente de una entrada CLK común. Cada salida de flip-flop alimenta un buffer de tres estados con salida de nivel activo alto. Todos los buffers de tres estados son habilitados mediante una entrada común, de nivel activo bajo OE_L (habilitación de salida).

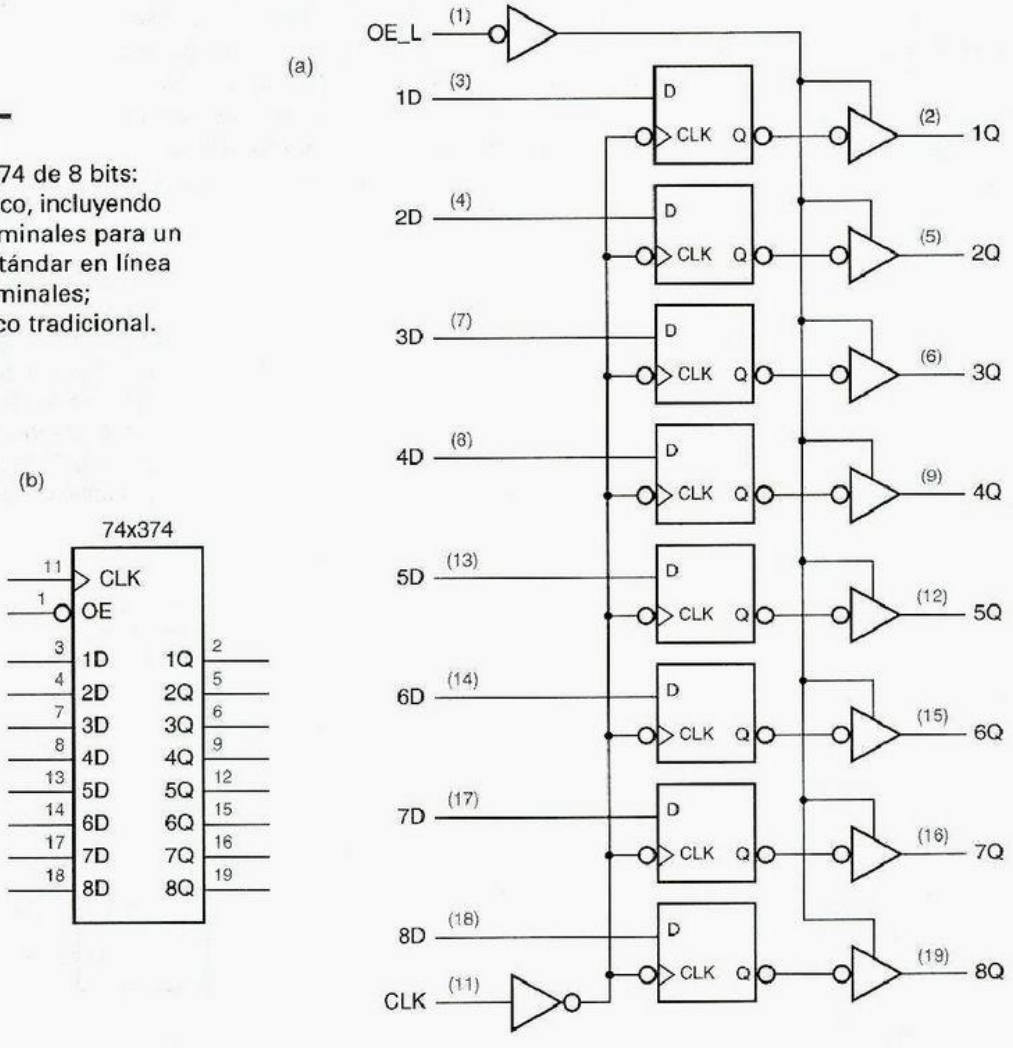


Figura 8-10
El registro 74x374 de 8 bits: a) diagrama lógico, incluyendo números de terminales para un encapsulado estándar en línea doble de 20 terminales; b) símbolo lógico tradicional.

Como en los otros registros que hemos estudiado, las entradas de control (CLK y OE_L) pasan por buffers de manera que presenten solamente una carga unitaria a un dispositivo que las controla.

Una variación del 74x374 es el *74x373*, cuyo símbolo se muestra en la figura 8-11. El '373 utiliza latchs D en lugar de flip-flops disparados por flanco. Por lo tanto, sus salidas siguen las entradas correspondientes siempre que C sea asertiva, y aseguran los últimos valores de entrada cuando C es negada. Otra variación es el *74x273*, mostrado en la figura 8-12. Este registro octal tiene salidas que no son de tres estados, y no tiene entrada OE_L; en su lugar emplea la terminal 1 para una entrada de borrado asincrónica CLR_L.

El *74x377*, cuyo símbolo se ilustra en la figura 8-13(a), es un registro disparado por flanco como el '374, pero no tiene salidas de tres estados. En su lugar, se utiliza la terminal 1 como una entrada de habilitación de reloj de nivel activo bajo EN_L. Si EN_L es asertiva (BAJO) en el flanco ascendente del reloj, entonces los flip-flops son cargados desde las entradas de datos; de otro modo, retienen sus valores presentes, como se muestra lógicamente (b).

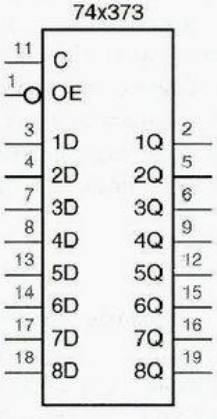


Figura 8-11 Símbolo lógico para el latch de 8 bits 74x373.

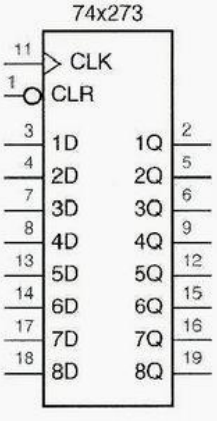
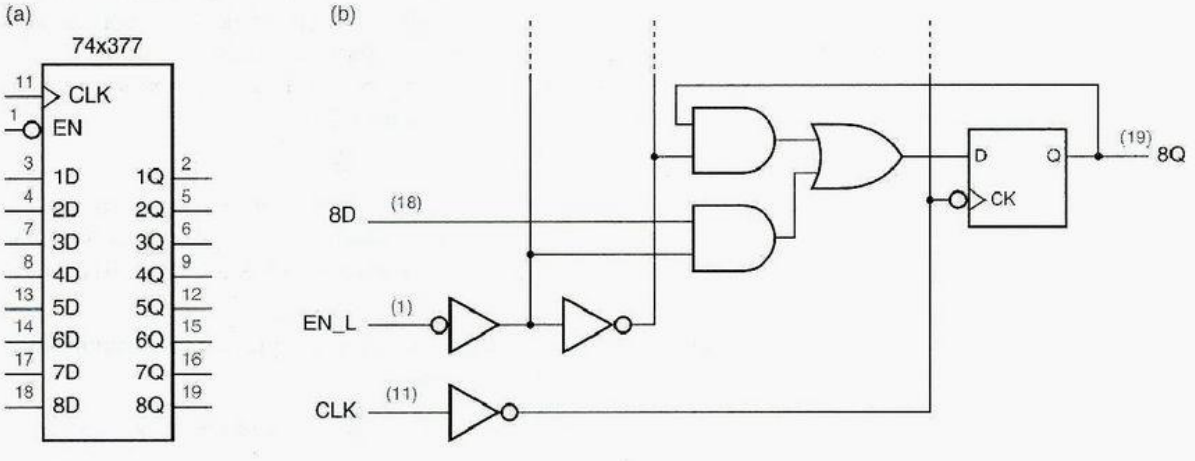


Figura 8-12 Símbolo lógico para el registro de 8 bits 74x273.

Figura 8-13 El registro de 8 bits 74x377 con habilitación de reloj: a) símbolo lógico; b) el comportamiento lógico de un bit.



Los encapsulados de montaje superficial con alto conteo de terminales soportan incluso registros controladores y transceptores más amplios. Los más comunes son los dispositivos de 16 bits, pero también existen dispositivos con 18 bits (para la paridad de byte) y 32 bits. También, los encapsulados más grandes pueden ofrecer más funciones de control, tales como borrado, habilitación de reloj, habilitaciones de salidas múltiples e inclusive una selección de comportamiento de latch vs., registro todo en un dispositivo.

8.2.6 Registros y latchs en ABEL y PLD

Como mostramos en la sección 7.11, los registros son muy fáciles de especificar en ABEL. Por ejemplo, la tabla 7-33 de la página 639 mostraba un programa ABEL para un registro de 8 bits con habilitación. Obviamente, ABEL permite que las funciones realizadas en las entradas D del registro sean personalizadas en casi cualquier forma que se desee, limitándose solamente por el número de entradas y términos de producto en el PLD objetivo. Describiremos los PLD secuenciales en la sección 8.3.

Con la mayoría de los PLD secuenciales pocas o más bien ninguna personalización puede aplicarse a una entrada de reloj del registro (por ejemplo, elección de polaridad) o a las entradas asincrónicas (por ejemplo, diferentes condiciones de preestablecimiento para bits diferentes). Sin embargo, ABEL proporciona la sintaxis adecuada para aplicar estas personalizaciones en dispositivos que los soporten, como se describió en la sección 7.11.1.

Muy pocos PLD tienen latchs integrados; los registros disparados por flanco son mucho más comunes y generalmente más útiles. Sin embargo, también se puede sintetizar un latch que utiliza lógica combinacional y retroalimentación. Por ejemplo, la ecuación de excitación para un latch S-R es

$$Q^* = S + R' \cdot Q$$

De este modo, se podría construir un latch S-R que utiliza solamente una salida combinacional de un PLD, empleando la ecuación ABEL "`Q = S # !R & Q.`" Además, las señales S y R anteriores podrían ser reemplazadas con funciones lógicas más complejas de las entradas del PLD, limitadas solamente por la disponibilidad de los términos de producto (siete por salida en un 16V8C o 16L8) para realizar la ecuación de excitación final. El lazo de retroalimentación puede ser creado sólo cuando Q es asignada a una terminal bidireccional (en un 16V8C o 16L8, terminales IO2-IO7, no O1 u O8). También, la terminal de salida debe ser continuamente habilitada de salida; de otro modo, el lazo de retroalimentación podría romperse y perderse el estado del latch.

Probablemente el latch más práctico a realizar con un PLD combinacional es un latch D. La ecuación de excitación básica para un latch D es

$$Q^* = C \cdot D + C' \cdot Q$$

Sin embargo, mostramos en la sección 7.10.1 que esta ecuación contiene un riesgo estático y que el circuito correspondiente no asegura los datos de manera confiable. Para construir un latch D confiable, debemos incluir un término de consenso en la ecuación de excitación:

$$Q^* = C \cdot D + C' \cdot Q + D \cdot Q$$

La entrada D en esta ecuación puede reemplazarse con una expresión más complicada, pero la estructura de la ecuación permanece igual:

$$Q^* = C \cdot \textit{expresión} + C' \cdot Q + \textit{expresión} \cdot Q$$

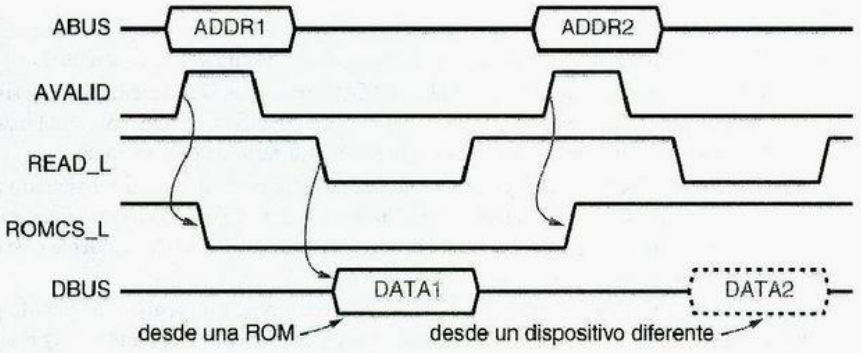


Figura 8-14
Diagrama de temporización para una operación de lectura de microprocesador.

También es posible utilizar una expresión más compleja para la entrada C, como mostramos en la sección 7.10.1. En cualquier caso, es muy importante para el término de consenso ser incluido en la realización del PLD. El compilador puede trabajar contra usted en este caso, puesto que su paso de minimización encontrará que el término de consenso es redundante y lo eliminará.

Algunas versiones del compilador ABEL permiten la eliminación de los términos de consenso al incluir una palabra clave "`retain`" en la lista de propiedades de la declaración `istype` para cualquier salida que no vaya a ser minimizada. En otras versiones, su única elección es desactivar la minimización para todo el diseño.

propiedad `retain`

Probablemente el uso más común de un latch basado en PLD es simultáneamente decodificar y asegurar direcciones a fin de seleccionar memoria y dispositivos de E/S en sistemas de microprocesadores. La figura 8-14 es un diagrama de temporización para esta función en un sistema típico. El microprocesador selecciona un dispositivo y una ubicación dentro del dispositivo colocando una dirección en su bus de direcciones (ABUS) y afirmando una señal de "dirección válida" (AVALID). Un breve tiempo después, establece una señal de lectura (READ_L) y el dispositivo seleccionado responde colocando los datos en el bus de datos (DBUS).

Note que la dirección no permanece válida en ABUS durante toda la operación. El protocolo del bus del microprocesador espera la dirección que será asegurada utilizando AVALID como una habilitación, después decodificada, como se muestra en la figura 8-15. El decodificador selecciona diferentes dispositivos para ser habilitados o "seleccionados en chip" de acuerdo con los bits de orden mayor de la dirección (los 12 bits de mayor orden en este ejemplo). Los bits de orden menor son empleados para direccionar localidades individuales de un dispositivo seleccionado.

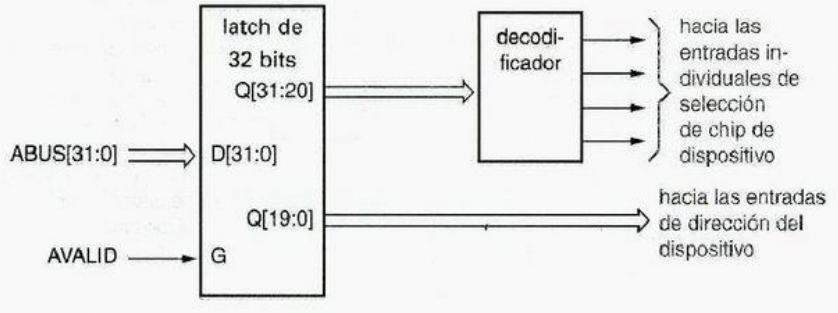


Figura 8-15
Circuito microprocesador de latch de dirección y decodificación.

¿POR QUÉ UN LATCH?

El protocolo del bus del microprocesador en la figura 8-14 suscita varias preguntas:

- ¿Por qué no conservar la dirección válida en ABUS para toda la operación? En un sistema real utilizando este protocolo, las funciones de ABUS y DBUS son combinadas (multiplexadas) en un bus de tres estados para ahorrar terminales y alambres.
- ¿Por qué no emplear AVALID como la entrada de reloj para un registro disparado por flanco positivo para capturar la dirección? No hay suficiente tiempo de establecimiento; en un sistema real, la dirección puede ser primero validada durante o ligeramente después del flanco ascendente de AVALID.
- Está bien, así que ¿por qué no usar AVALID para temporizar un registro disparado por flanco negativo? Esto funciona, pero las salidas aseguradas quedan disponibles muy pronto; los valores válidos en ABUS fluyen a través de un latch inmediatamente, sin esperar el flanco de caída del reloj. Esto relaja los requerimientos de tiempo de acceso de memorias y otros dispositivos controlados por las salidas aseguradas.

Utilizando un PLD las funciones de latch y decodificación para los bits de mayor orden pueden combinarse en un solo dispositivo, produciendo los diagramas de bloques en la figura 8-16. En comparación con la figura 8-15, el "decodificador de latch" ahorra dispositivos y terminales produciendo una salida de selección de chip válida con mayor rapidez (vea el ejercicio 8.9).

La tabla 8-2 es un programa ABEL para el decodificador de cerrojo. Puesto que funciona solamente en los bits de mayor orden `ABUS[31..20]`, puede decodificar direcciones solamente en 1 Mbyte o segmentos mayores (2^{20} = 1 M). Una memoria sólo de lectura (ROM) se localiza en el segmento de 1 Mbyte más alto, en las direcciones `0xfff00000-0xffffffff`, y es seleccionada por `ROMCS`. Tres bancos de 16 Mbytes de memoria de lectura/escritura (RAM) se localizan en las direcciones inferiores, comenzando en las direcciones `0x00000000`, `0x01000000`, y `0x02000000`, respectivamente. Note cómo se utilizan los elementos sin importancia en las definiciones de los intervalos de direcciones del banco de RAM para decodificar un segmento mayor de 1 Mbyte. También son posibles otros enfoques a estas definiciones (por ejemplo, vea el ejercicio 8.1.).

Las ecuaciones en la tala 8-2 para las salidas de selección de chip siguen la plantilla de latch D que dimos en la página 674. Las expresiones que seleccionan un dispositivo, tal como "`ABUS==ROM`,", generan cada una un término de producto simple, y cada ecuación genera tres términos de producto. Advierta el uso de la propiedad "`retain`" en las declaraciones de terminales para prevenir al compilador de optimizar fuera de los términos de consenso.

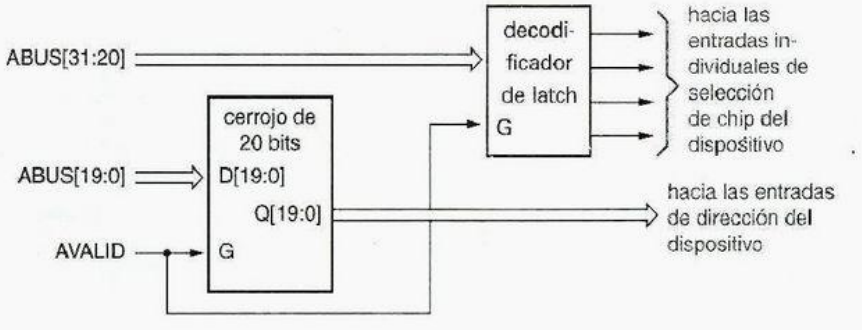


Figura 8-16
Uso de un circuito combinado decodificador y de cerrojo de dirección.

Tabla 8-2
Programa ABEL para un decodificador de dirección de latch.

```
module latchdec
title 'Latching Microprocessor Address Decoder'

" Inputs
AVALID, ABUS31..ABUS20          pin;
" Latched and decoded outputs
ROMCS, RAMCS0, RAMCS1, RAMCS2   pin istype 'com,retain';

ABUS = [ABUS31..ABUS20];
ROM = ^hFFF;
RAMBANK0 = [0,0,0,0,0,0,0,0,.X.,.X.,.X.,.X.];
RAMBANK1 = [0,0,0,0,0,0,0,1,.X.,.X.,.X.,.X.];
RAMBANK2 = [0,0,0,0,0,0,1,0,.X.,.X.,.X.,.X.];

equations

ROMCS  = AVALID & (ABUS==ROM) # !AVALID & ROMCS
       # (ABUS==ROM) & ROMCS;
RAMCS0 = AVALID & (ABUS==RAMBANK0) # !AVALID & RAMCS0
       # (ABUS==RAMBANK0) & RAMCS0;
RAMCS1 = AVALID & (ABUS==RAMBANK1) # !AVALID & RAMCS1
       # (ABUS==RAMBANK1) & RAMCS1;
RAMCS2 = AVALID & (ABUS==RAMBANK2) # !AVALID & RAMCS2
       # (ABUS==RAMBANK2) & RAMCS2;

end latchdec
```

Después de ver lo fácil que es construir latchs S-R y D con la ayuda de los PLD combinacionales, usted desearía ir más allá e intentar construir un flip-flop D disparado por flanco. Aunque esto es posible, es costoso porque un flip-flop disparado por flanco tiene cuatro estados internos y de este modo dos lazos de retroalimentación, consumiendo dos salidas del PLD. Además, los tiempos de establecimiento y retención y retardos de propagación de un flip-flop así serían bastante deficientes en comparación con los correspondientes a un flip-flop discreto de la misma tecnología. Finalmente, como lo discutimos en la sección 7.10.6, las tablas de flujo de cualquier flip-flop disparado por flanco contienen riesgos esenciales, que solamente se pueden eliminar al controlar retardos de trayectoria, difícil basado en PLD.

8.2.7 Registros y latchs en VHDL

Los circuitos de registro latch se pueden especificar utilizando VHDL estructural. Por ejemplo, la tabla 8-3 es un programa VHDL estructural correspondiente al circuito de latch D de la figura 7-12 en la página 539. Sin embargo, escribir programas estructurales no es realmente nuestra motivación para emplear VHDL; nuestro objetivo es utilizar programas de comportamiento para modelar el funcionamiento de los circuitos más intuitivamente.

La tabla 8-4 es una arquitectura de comportamiento basada en proceso para el latch D que requiere, en efecto, solamente una línea de código para describir el comportamiento

Tabla 8-3 Programa estructural VHDL para el latch D en la figura 7-12.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdlatch is
  port (D, C: in STD_LOGIC;
        Q, QN: buffer STD_LOGIC );
end Vdlatch;

architecture Vdlatch_s of Vdlatch is
  signal DN, SN, RN: STD_LOGIC;
  component inv port (I: in STD_LOGIC; O: out STD_LOGIC ); end component;
  component nand2b port (I0, I1: in STD_LOGIC; O: buffer STD_LOGIC ); end component;
begin
  U1: inv port map (D,DN);
  U2: nand2b port map (D,C,SN);
  U3: nand2b port map (C,DN,RN);
  U4: nand2b port map (SN,QN,Q);
  U5: nand2b port map (Q,RN,QN);
end Vdlatch_s;
```

Tabla 8-4 Arquitectura de comportamiento VHDL para un latch D.

```
architecture Vdlatch_b of Vdlatch is
begin
process(C, D, Q)
  begin
    if (C='1') then Q <= D; end if;
    QN <= not Q;
  end process;
end Vdlatch_b;
```

latch deducido

del latch. Advierta que el compilador VHDL "infiere" o "deduce" un latch de esta descripción, puesto que el código no dice qué hacer si C no es 1, el compilador crea *latch deducido* para retener el valor de Q entre las invocaciones del proceso. En general, un compilador VHDL deduce un latch para una señal a la que se asigna un valor en una instrucción `if` o `case` si no se incorporan todas las combinaciones de entrada.

atributo event

Con la finalidad de describir el comportamiento disparado por flanco de los flip-flops, necesitamos emplear uno de los atributos de señal predefinidos de VHDL, el atributo `event`. Si "SIG" es un nombre de señal, entonces la construcción "SIG" *event*" regresa el valor *true* en cualquier tiempo delta en que SIG cambia un valor a otro, y a `false` en caso contrario.

Utilizando el atributo `event`, podemos modelar un flip-flop disparado por flanco positivo como se muestra en la tabla 8-6. En la instrucción `IF`, "`CLK'event`" regresa `true` en cualquier flanco de reloj, "`CLK='1'`" asegura que D es asignada a Q solamente en un flanco *ascendente*. Note que la lista de sensibilidad del proceso incluye solamente CLK; los cambios D en otros instantes no son relevantes en este modelo funcional.

AFICIONADOS AL TEMA

Note que en la tabla 8-3 definimos el tipo de Q y QN como buffer en vez de out, puesto que estas señales se utilizan como entradas además de cómo salidas en la definición de arquitectura. Entonces teníamos que definir una compuerta NAND especial de 2 entradas nand2b con tipo de salida "buffer" para evitar tener un desequilibrio de tipos (out versus buffer) en las instancias de componente para U4 y U5. Alternativamente, podríamos utilizar las señales internas para resolver el problema, como se muestra en la tabla 8-5. Como usted sabe ahora, VHDL tiene muchas maneras diferentes de expresar la misma cosa.

Tabla 8-5 Programa estructural VHDL alternativo para el latch D en la figura 7-12.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdlatch is
  port (D, C: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end Vdlatch;

architecture Vdlatch_s2 of Vdlatch is
  signal DN, SN, RN, IQ, IQN: STD_LOGIC;
  component inv port (I: in STD_LOGIC; O: out STD_LOGIC ); end component;
  component nand2 port (I0, I1: in STD_LOGIC; O: out STD_LOGIC ); end component;
begin
  U1: inv port map (D,DN);
  U2: nand2 port map (D,C,SN);
  U3: nand2 port map (C,DN,RN);
  U4: nand2 port map (SN,IQN,IQ);
  U5: nand2 port map (IQ,RN,IQN);
  Q <= IQ; QN <= IQN;
end Vdlatch_s2;
```

Tabla 8-6 Modelo de comportamiento VHDL de un flip-flop D disparado por flanco.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdff is
  port (D, CLK: in STD_LOGIC;
        Q: out STD_LOGIC );
end Vdff;

architecture Vdff_b of Vdff is
begin
process(CLK)
  begin
    if (CLK'event and CLK='1') then Q <= D; end if;
  end process;
end Vdff_b;
```

Tabla 8-7 Modelo VHDL para un flip-flop D tipo 74x74 con prestablecimiento y borrado.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdff74 is
  port (D, CLK, PR_L, CLR_L: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end Vdff74;

architecture Vdff74_b of Vdff74 is
signal PR, CLR: STD_LOGIC;
begin
process(CLR_L, CLR, PR_L, PR, CLK)
  begin
    PR <= not PR_L; CLR <= not CLR_L;
    if (CLR and PR) = '1' then Q <= '0'; QN <= '0';
    elsif CLR = '1' then Q <= '0'; QN <= '1';
    elsif PR = '1' then Q <= '1'; QN <= '0';
    elsif (CLK'event and CLK='1') then Q <= D; QN <= not D;
    end if;
  end process;
end Vdff74_b;
```

Tabla 8-8 Modelo VHDL de un registro de 16 bits con muchas características.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vreg16 is
  port (CLK, CLKEN, OE_L, CLR_L: in STD_LOGIC;
        D: in STD_LOGIC_VECTOR(1 to 16);       -- Bus de entrada
        Q: out STD_LOGIC_VECTOR (1 to 16); -- Bus de salida (tres estados)
end Vreg16;

arquitecture Vreg16 of Vreg16 is
signal CLR, OE: STD_LOGIC; -- Versiones de señales en estado activo alto
signal IQ: STD_LOGIC_VECTOR(1 to 16); señales Q internas
begin
process (CLK, CLR_L, CLR, OE_L, OE, IQ)
  begin
    CLR <= not CLR_L; OE <= not OE_L;
    if (CLR = '1') then IQ <= (others => '0');
    elsif (CLK' event and CLK'1') then
       if (CLKEN='1') then IQ <= D; end if;
    end if;
    if OE = '1' then Q <= IQ;
     else Q <= (others => 'Z'); end if;
  end process;
end Vreg16;
```

El modelo de flip-flop D puede ser aumentado para incluir entradas asincrónicas y una salida complementada como en el flip-flop discreto 74x74, como se muestra en la tabla 8-7. Este modelo funcional más detallado muestra el comportamiento no complementario de las salidas Q y QN cuando el prestablecimiento y el borrado son asertivos de manera simultánea. Sin embargo, no incluye comportamiento de temporización tal como retardo de propagación y tiempos de establecimiento y retención, los cuales se encuentran más allá del temario de VHDL que se cubre en este libro.

Los registros más grandes pueden modelarse, naturalmente, mediante la definición de las entradas y salidas de datos como vectores, y también pueden incluirse funciones adicionales, la tabla 8-8 modela un registro de 16 bits con salidas de tres estados y entradas de habilitación de reloj, habilitación de salida y de borrado. Un vector de señal interna IQ se emplea para mantener las salidas del flip-flop, y se definen y habilitan salidas de tres estados como la sección 5.6.4.

RESTRICCIONES DE SÍNTESIS

En la tabla 8-8, la primera instrucción `elsif` teóricamente podría haber incluido todas las condiciones necesarias para asignar D a IQ. Es decir, podría haber leído "`elsif (CLK'event) and (CLK='1') and (CLKEN='1') then ...`" en vez de emplear una instrucción `if` anidada para verificar CLKEN. Sin embargo, fue escrita como se muestra por una razón pragmática.

Solamente un subconjunto del lenguaje VHDL puede ser sintetizado por el compilador de VHDL que fue empleado para preparar este capítulo; esto es verdadero para cualquier compilador VHDL en la actualidad. En particular, el uso del atributo "`event`" está limitado a la forma mostrada en el ejemplo, y unas cuantas más, para detectar el comportamiento disparado por flanco simple. Esto llega a mapearse en flip-flops D disparados por flanco durante la síntesis. La instrucción IF anidada que verifica CLKEN en el ejemplo conduce a la síntesis de lógica de multiplexor en las entradas D de estos flip-flops.

8.3 PLD secuenciales

8.3.1 PLD secuenciales bipolares

PAL16R8

El *PAL16R8* que se muestra en la figura 8-17 es representativo de la primera generación de los PLD secuenciales, que utilizaban tecnología bipolar (TTL). Este dispositivo tiene ocho entradas primarias, ocho salidas y entradas comunes de reloj y habilitación de salida, y cabe en un encapsulado de 20 terminales.

El arreglo AND-OR del PAL16R8 es exactamente el mismo que el encontrado en el PLD combinacional PAL16L8. Sin embargo, el PAL16R8 tiene flip-flops D disparados por flanco entre el arreglo AND-OR y sus ocho salidas, O1-O8. Todos los flip-flops se encuentran conectados a una entrada de reloj común. CLK, y cambian estado en el flanco ascendente del reloj. Cada flip-flop pasa a una terminal de salida a través de un buffer de tres estados; los buffers tienen una señal común de habilitación de salida, OE_L. Advierta que, como las terminales de salida combinacional de un PAL16L8, las

terminales de salida registradas del PAL16R8 contienen el complemento de la señal producida por el arreglo AND-OR.

Las entradas posibles para el arreglo AND-OR del PAL16R8 son ocho entradas primarias (I1-I8) y las ocho salidas de flip-flop D. La conexión de las salidas del flip-flop D en el arreglo AND-OR hace fácil diseñar registros de corrimiento, contadores y máquinas de estado en general. A diferencia de las salidas combinacionales del PAL16L8, las salidas del flip-flop D del PAL16R8 se encuentran disponibles para el arreglo AND-OR ya sea que estén o no habilitados los controladores de tres estados O1-O8. De este modo, los flip-flops internos pueden ir a un estado siguiente que es una función del estado actual aun cuando las salidas O1-O8 se encuentren deshabilitadas.

Muchas aplicaciones requieren de salidas PLD tanto combinacionales como secuenciales. Los fabricantes de los PLD bipolares abordaron esta necesidad proporcionando unas cuantas variantes del PAL16R8 que omitían los flip-flops D en algunas terminales de salida, y en su lugar proporcionaban capacidad de entrada y salida idéntica a la de las terminales bidireccionales del PAL16L8. Por ejemplo, la figura 8-18 es el diagrama lógico del *PAL16R6*, el cual solamente tiene seis salidas registradas. Dos terminales, IO1 e IO8, son bidireccionales, y sirven tanto como entradas y como salidas combinacionales con habilitaciones de tres estados por separado, justo igual que las terminales bidireccionales del PAL16R8. De esta forma, las posibles entradas para el arreglo AND-OR del PAL16R6 son las ocho entradas primarias (I1–I8), las seis salidas del flip-flop D, y las dos terminales bidireccionales (IO1, IO8).

PAL16R6

PAL16L8
PAL16R4
PAL16R8
PAL20L8
PAL20R4

La tabla 8-9 muestra ocho PLD bipolares estándar con diferentes números y tipos de entradas y salidas. Todas las partes PAL16xx en la tabla utilizan el mismo arreglo AND-OR, donde cada salida tiene ocho compuertas AND, cada una con 16 variables y sus complementos como entradas posibles. Las partes PAL20xx emplean un arreglo

Tabla 8-9 Características de los PLD bipolares estándar.

			Entradas al arreglo AND			
Número de parte	***Terminales del encapsulado***	***Entradas compuerta AND***	***Entradas primarias***	***Salidas combinacionales bidireccionales***	***Salidas registradas***	***Salidas combinacionales***
PAL16L8	20	16	10	6	0	2
PAL16R4	20	16	8	4	4	0
PAL16R6	20	16	8	2	6	0
PAL16R8	20	16	8	0	8	0
PAL20L8	24	20	14	6	0	2
PAL20R4	24	20	12	4	4	0
PAL20R6	24	20	12	2	6	0
PAL20R8	24	20	12	0	8	0

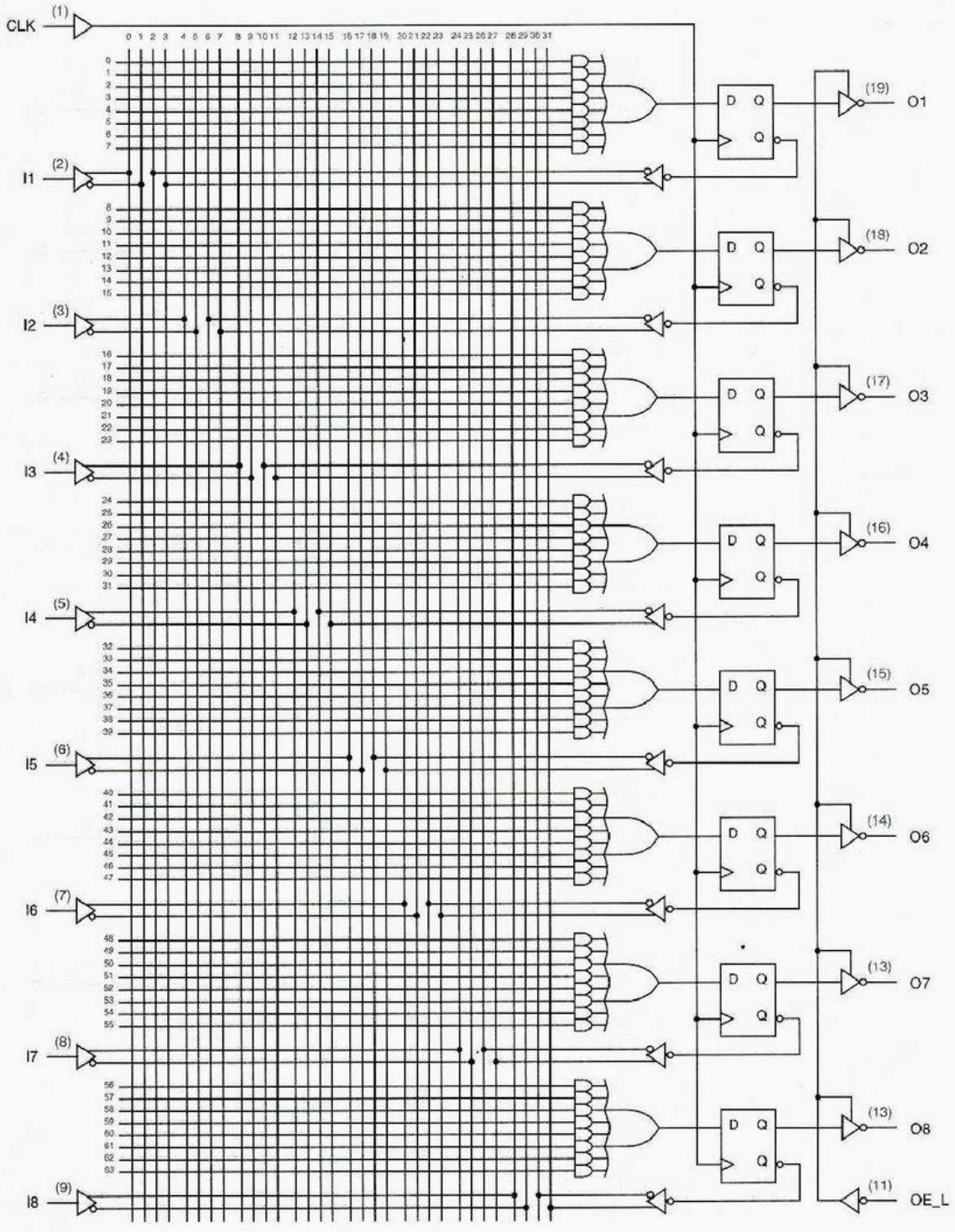


Figura 8-17 Diagrama lógico del PAL16R8.

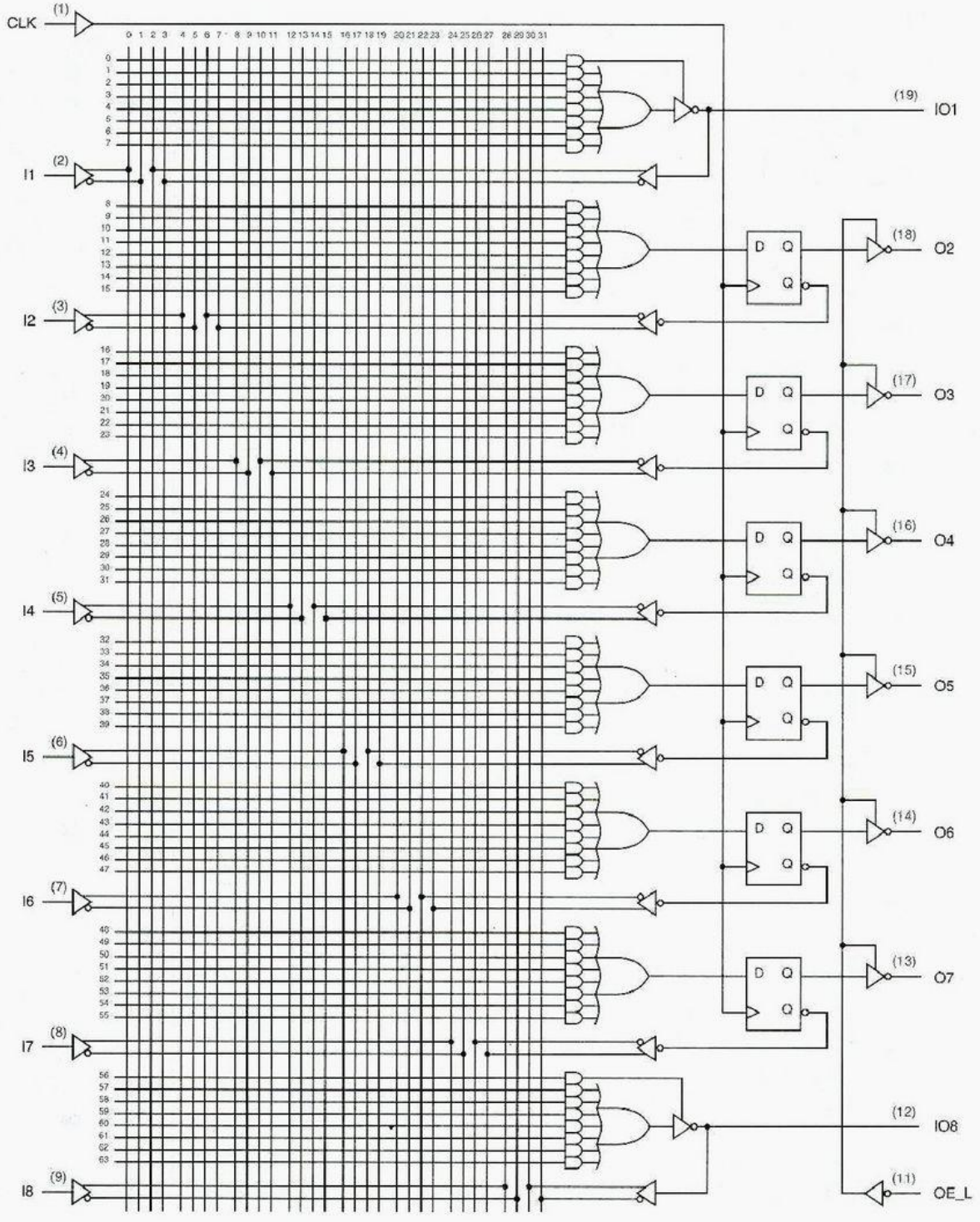


Figura 8-18 Diagrama lógico del PAL16R6.

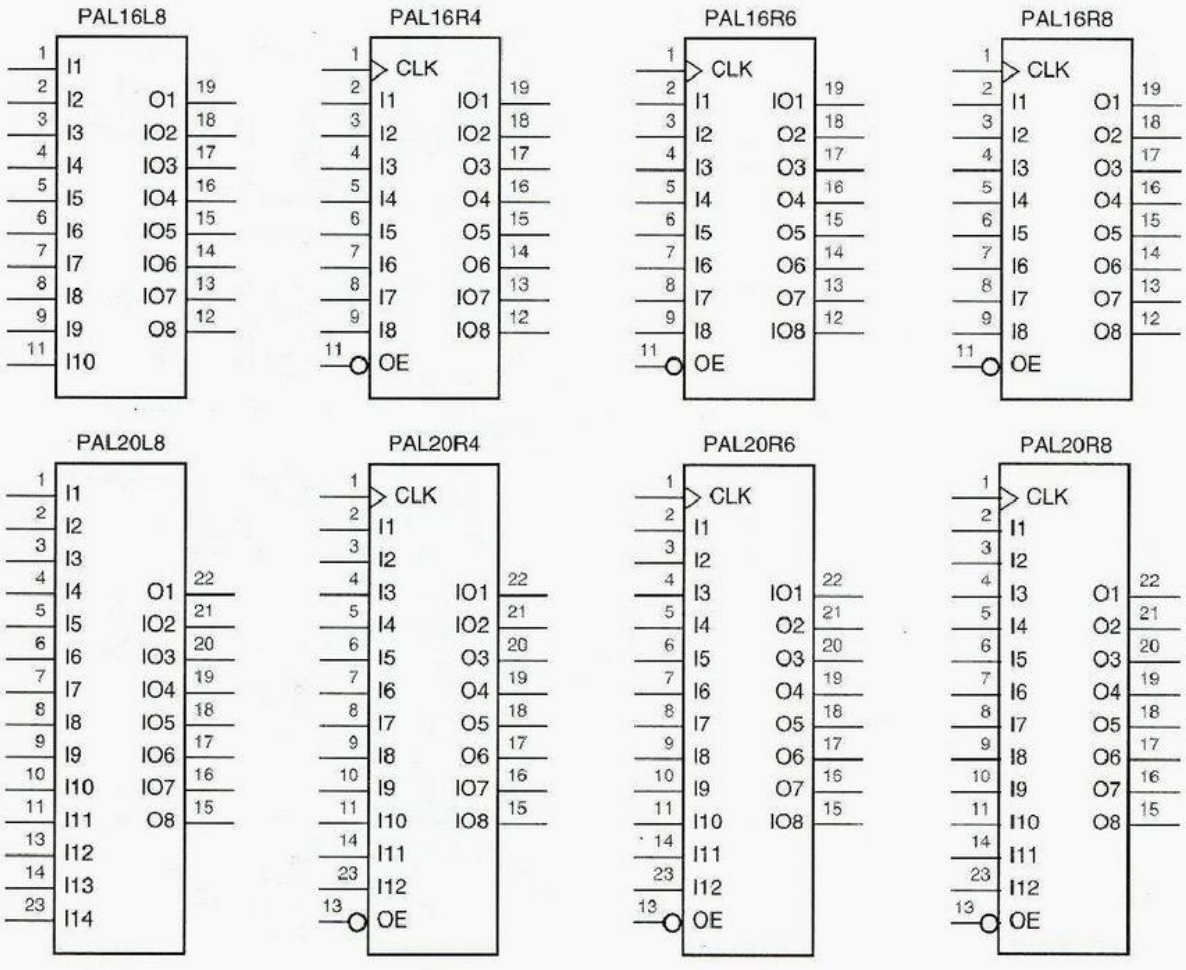


Figura 8-19 Símbolos lógicos para los PLD secuenciales y combinacionales bipolares.

AND-OR con 20 variables y sus complementos como posibles entradas. La figura 8-19 ilustra los símbolos lógicos para todos los PLD en la tabla. *PAL20R6* *PAL20R8*

8.3.2 Dispositivos GAL secuenciales

El PLD eléctricamente borrable GAL16V8 fue presentado en la sección 5.3.3. Dos fusibles de "control de arquitectura" se utilizan para seleccionar entre tres configuraciones básicas de este dispositivo. En la sección 5.3.3 describimos la configuración *16V8C* ("compleja"), que se muestra en la figura 5-27, una estructura similar a la del PAL combinacional bipolar, el PAL16L8. La configuración *16V8S* ("simple") proporciona una capacidad lógica combinacional ligeramente diferente (vea el cuadro en la página 687). *16V8C* *16V8S*

La tercera configuración, la *16V8R*, permite incluir un flip-flop en alguna o en todas las salidas. La figura 8-20 ilustra la estructura del dispositivo cuando todas las *16V8R*

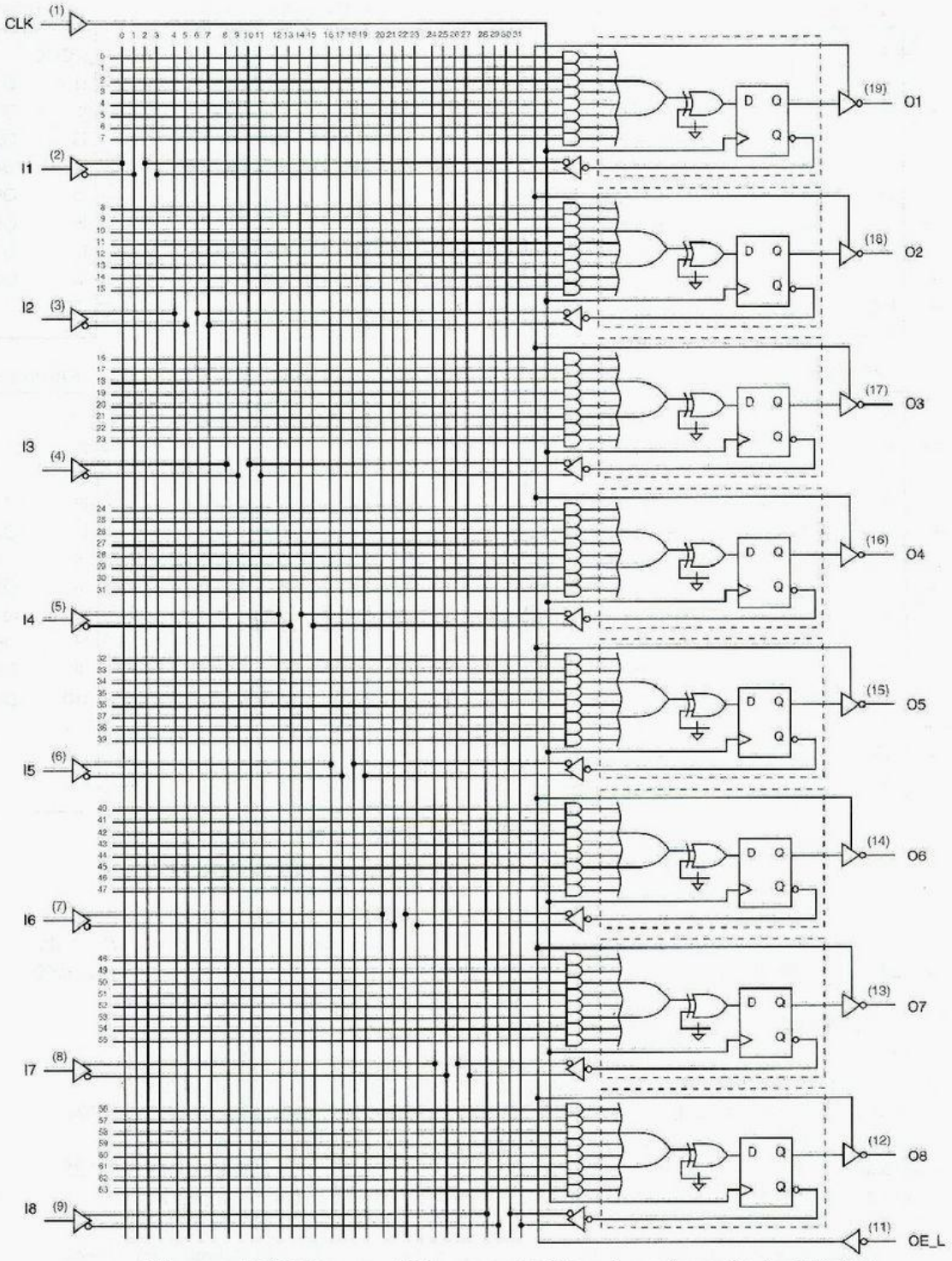


Figura 8-20 Diagrama lógico para el 16V8 en la configuración "registrada".

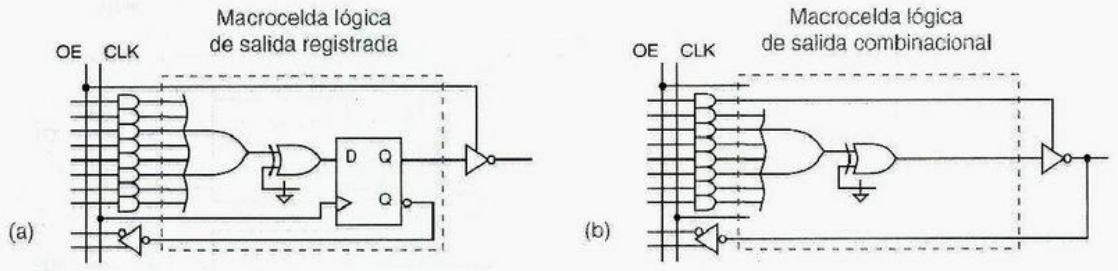


Figura 8-21 Macroceldas lógicas de salida para el 16V8R: (a) registradas; (b) combinacionales.

salidas están equipadas con flip-flops (dispositivos biestables). Note que todos los flip-flops están controlados por una señal de reloj común en la terminal 1, como en los dispositivos bipolares de la sección anterior. De manera similar, todos los buffers de salida están controlados por una señal común de habilitación de salida en la terminal 11.

La circuitería dentro de cada cuadro con línea punteada en la figura 8-20 se denomina una *macrocelda lógica de salida*. El 16V8R es mucho más flexible que un PAL16R8 porque cada macrocelda puede ser configurada de manera individual para evitar el flip-flop; es decir, para producir una salida combinacional. La figura 8-21 ilustra las dos configuraciones de macrocelda posibles en el 16V8R; (a) es registrada y (b) es combinacional. De este modo, es posible programar el dispositivo para tener cualquier conjunto de salidas registradas y combinacionales, hasta un total de ocho.

macrocelda lógica de salida

El *20V8* es semejante al 16V8 pero viene en un encapsulado de 24 terminales con cuatro terminales extra solamente para entrada. Cada compuerta AND en el 20V8 tiene 20 entradas, de ahí el "20" que viene en "20V8".

20V8

EL "SIMPLE" 16V8S

La "simple" configuración 16V8S del GAL16V8 no se utiliza con frecuencia, porque sus capacidades son principalmente un subconjunto de las del 16V8C. En vez de un término AND, el 16V8S emplea un fusible por salida para controlar si los buffers de salida se habilitan. Es decir, cada terminal de salida puede ser programada ya sea para estar siempre habilitada o para estar siempre deshabilitada (excepto las terminales 15 y 16, que siempre se encuentran habilitadas). Todas las terminales de salida (exceptuando la 15 y 16) se encuentran disponibles como entradas al arreglo AND sin tener en cuenta si el buffer de salida está habilitado.

La única ventaja de un 16V8S en comparación con un 16V8C es que tiene ocho, no siete, términos AND como entradas a la compuerta OR en cada salida. La arquitectura del 16V8S fue diseñada principalmente para la emulación de ciertos dispositivos PAL bipolares ahora obsoletos, algunos de los cuales tenía ocho términos de producto por salida o bien tenían entradas en las terminales 12 y 19, que no eran entradas en la configuración 16V8C. Con una adecuada programación, el 16V8S se puede utilizar como un reemplazo compatible (de terminal a terminal) para estos dispositivos, entre los que se incluyen el PAL10H8, PAL12H6, PAL14H4, PAL16H2, PAL10L8, PAL12L6, PAL14L4 y PAL16L2.

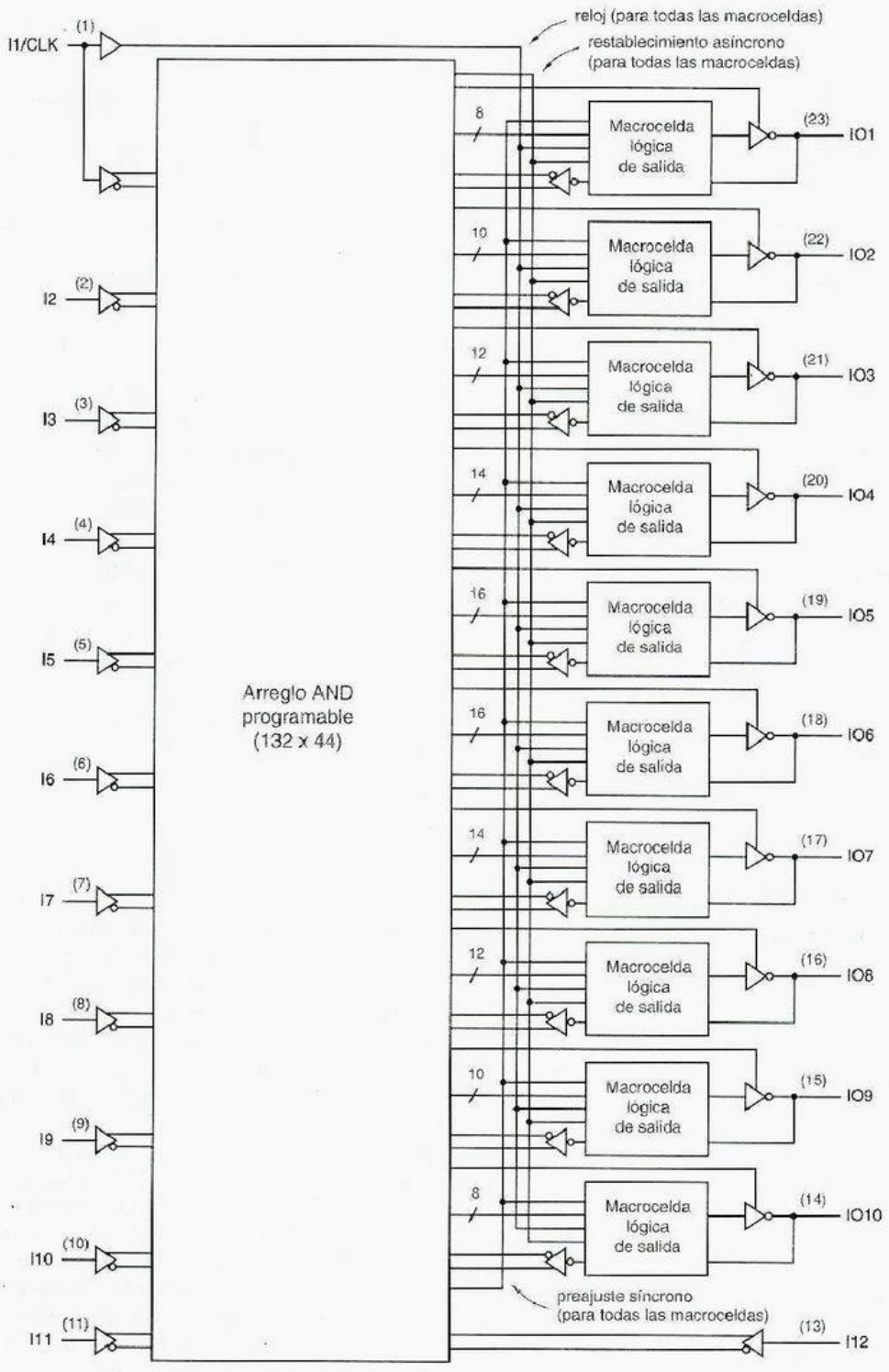


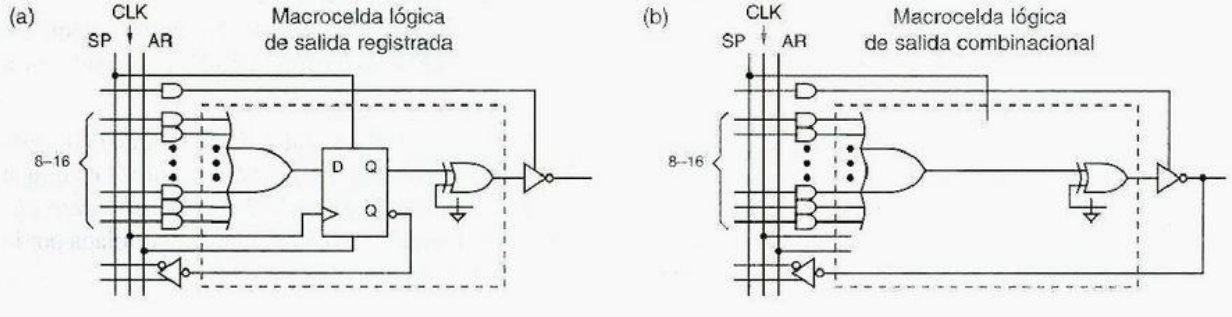
Figura 8-22
Diagrama lógico para el 22V10.

22V10

El *22V10*, cuya estructura básica se ilustra en la figura 8-22, también viene en un encapsulado de 24 terminales pero es algo más flexible que el 20V8. El 22V10 no tiene bits de "control de arquitectura" como los del 16V8 y el 20V8, pero puede realizar cualquier función que sea realizable con un 20V8, y más. Tiene más términos de producto, dos entradas más de propósito general, y mejor control de habilitación de salida que el 20V8. Las diferencias clave se resumen a continuación:

- Cada macrocelda lógica de salida es configurable para tener un registro o no, como en la arquitectura 20V8R. Sin embargo, las macroceldas son diferentes de las del 16V8 y 20V8, como se muestra en la figura 8-23.
- Un término de producto simple controla el buffer de salida, sin tener en cuenta si se selecciona la configuración registrada o combinacional para una macrocelda.
- Cada salida tiene al menos ocho términos de producto disponibles, sin tener en cuenta si se selecciona la configuración registrada o combinacional. Todavía más términos de producto están disponibles en las terminales internas, con 16 disponibles en cada una de las dos terminales más internas. ("Más internas" se refiere al lado derecho de la figura 8-22, que también satisface el arreglo de estas terminales en un encapsulado en línea doble de 24 terminales.)
- La señal de reloj en la terminal 1 también se encuentra disponible como una entrada combinacional para cualquier término de producto.
- Un término de producto simple se encuentra disponible para generar una señal de restablecimiento asincrónica, global, que reestablece todos los flip-flops internos a 0.
- Un término de producto simple se encuentra disponible para generar una señal de restablecimiento asincrónica, global, que establece todos los flip-flops internos a 1 en el flanco ascendente del reloj.
- Como el 16V8 y el 20V8, el 22V10 tiene polaridad de salida programable. Sin embargo, en la configuración registrada, el cambio de polaridad se hace en la salida, más que en la entrada, del flip-flop tipo D. Esto afecta los detalles de la programación cuando la polaridad se cambia pero no afecta la capacidad total del 22V10 (es decir, si una función dada puede ser realizada). De hecho, la diferencia en la ubicación del cambio de polaridad es transparente cuando se utiliza un lenguaje de programación PLD como ABEL.

Figura 8-23 Macroceldas lógicas de salida para el 22V10: (a) registradas; (b) combinacionales.



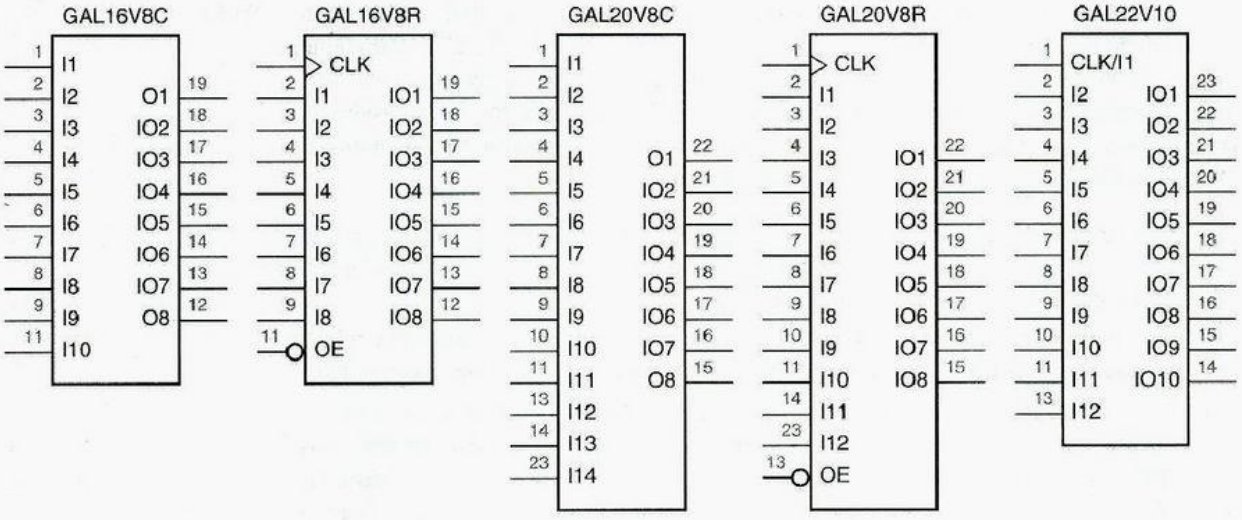


Figura 8-24 Símbolos lógicos para dispositivos populares GAL.

Durante la mayor parte de la década de los noventa, el 16V8, el 20V8 y el 22V10 fueron los PLD más populares y económicos (consulte el cuadro de la página 693). La figura 8-24 muestra símbolos lógicos genéricos para estos tres dispositivos. La mayor parte de los ejemplos en el resto de este capítulo pueden ajustarse dentro del más pequeño de los tres dispositivos, el 16V8.

¿PALS? ¿GALS? Lattice Semiconductor presentó los dispositivos GAL, incluyendo el GAL16V8 y el GAL20V8 a mediados de la década de 1980. Posteriormente Advanced Micro Devices profundizó en el asunto con un dispositivo de terminales compatibles al cual denominaron el PALCE16V8 ("C" es por CMOS, "E" es por "erasable": borrable en inglés). Otros fabricantes hacen también dispositivos compatibles con numeración diferente. Más que conseguir ponerse al día en los detalles de los diferentes nombres de los fabricantes, en este capítulo haremos referencia generalmente a los dispositivos GAL comúnmente empleados con sus nombres genéricos, 16V8, 20V8 y 22V10.

8.3.3 Especificaciones de temporización del PLD

Diversos parámetros de temporización son especificados para los PLD combinacionales y secuenciales. Los más importantes se ilustran en la figura 8-25 y se explican a continuación.

t_{PD}

entrada de retroalimentación

t_{PD} Este parámetro se aplica a las salidas combinacionales. Es el retardo de propagación desde una terminal de entrada primaria, terminal bidireccional o entrada de "retroalimentación" hasta la salida combinacional. Una *entrada de retroalimentación* es una entrada interna del arreglo AND-OR que es controlada por la salida registrada de una macrocelda interna.

t_{CO} Este parámetro se aplica a salidas registradas. Es el retardo de propagación desde el flanco ascendente de CLK hasta la salida primaria. t_{CO}

t_{CF} Este parámetro también se aplica a salidas registradas. Es el retardo de propagación desde el flanco ascendente de CLK hasta la salida registrada de una macrocelda que se conecta de vuelta a una entrada de retroalimentación. Si se especifica, t_{CF} es normalmente menor que t_{CO}. Sin embargo, algunos fabricantes no especifican t_{CF}, en cuyo caso se debe suponer que $t_{CF} = t_{CO}$. t_{CF}

t_{SU} Este parámetro se aplica a entradas primaria, bidireccional y de retroalimentación que se propagan a las entradas D de los flip-flops. Es el tiempo de establecimiento durante el cual la señal de entrada debe estar estable antes del flanco ascendente de CLK. t_{SU}

t_H Este parámetro también se aplica a señales que se propagan hacia las entradas D de los flip-flops. Es el tiempo de retención durante el cual la señal de entrada debe estar estable después del flanco ascendente de CLK. t_H

f_{max} Este parámetro se aplica a la operación temporizada. Es la frecuencia más alta a la cual el PLD puede funcionar confiablemente y es el recíproco del mínimo periodo de reloj. Se pueden obtener dos versiones de este parámetro a partir de las especificaciones anteriores, dependiendo de si el dispositivo se encuentra funcionando con retroalimentación externa o con retroalimentación interna. f_{max}

La *retroalimentación externa* se refiere a un circuito en el cual una salida PLD registrada se conecta a la entrada de otro PLD registrado con temporización similar; para un funcionamiento apropiado, la suma de t_{CO} para el primer PLD y t_{SU} para el segundo no deben exceder el periodo de reloj. *retroalimentación externa*

La *retroalimentación interna* se refiere a un circuito en el cual una salida PLD registrada se retroalimenta a un registro en el mismo PLD; en este caso, la suma de t_{CF} y t_{SU} no debe exceder el periodo del reloj. *retroalimentación interna*

Cada uno de los PLD que hemos descrito en secciones anteriores se encuentra disponible en varios grados diferentes de velocidad. El grado de velocidad se indica habitualmente mediante un sufijo en el número de parte, tal como "16V8-10"; el sufijo por lo regular se refiere a la especificación t_{PD}, en nanosegundos. La tabla 8-10 muestra

Figura 8-25 Parámetros de tiempo de PLD.

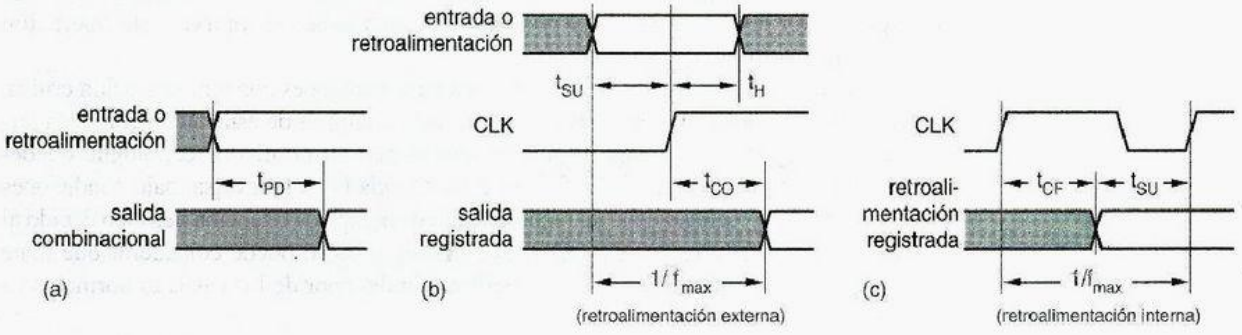


Tabla 8-10 Especificaciones de tiempo, en nanosegundos, de PLD CMOS y bipolares populares.

Números de parte	*Sufijo*	t_{PD}	t_{CO}	t_{CF}	t_{SU}	t_H
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-5	5	4	–	4.5	0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-7	7.5	6.5	–	7	0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-10	10	8	–	10	0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	B	15	12	–	15	0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	B-2	25	15	–	25	0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	A	25	15	–	25	0
PALCE16V8, PALCE20V8	-5	5	4	–	3	0
GAL16V8, GAL20V8	-7	7.5	5	3	5	0
GAL16V8, GAL20V8	-10	10	7.5	6	7.5	0
GAL16V8, GAL20V8	-15	15	10	8	12	0
GAL16V8, GAL20V8	-25	25	12	10	15	0
PALCE22V10	-5	5	4	–	3	0
PALCE22V10	-7	7.5	4.5	–	4.5	0
GAL22V10	-10	10	7	2.5	7	0
GAL22V10	-15	15	8	2.5	10	0
GAL22V10	-25	25	15	13	15	0

la temporización de varios PLD CMOS y bipolares populares. Advierta que solamente la columna t_{PD} se aplica a las salidas combinacionales de un dispositivo, mientras que las últimas cuatro columnas se aplican a salidas registradas. Todas las especificaciones de tiempo son los números en el peor de los casos sobre el intervalo de operación comercial.

Cuando se utilizan los PLD secuenciales en aplicaciones con temporización crítica, es importante recordar que normalmente ellos tienen tiempos de establecimiento más largos que los registros discretos disparados por flanco en la misma tecnología, obedeciendo al retardo del arreglo AND-OR en cada entrada D. A la inversa, bajo condiciones típicas, un PLD en realidad tiene un requerimiento de tiempo de retención negativo debido al retardo a través del arreglo AND-OR. Sin embargo, no se puede considerar que tiene un tiempo de retención negativo: la especificación del peor de los casos es normalmente cero.

¿CUÁNTO CUESTA?

Una vez que se han entendido las capacidades de los diferentes PLD, uno se podría preguntar, "¿por qué no sólo utilizar el PLD más capaz disponible?" Por ejemplo, incluso si un circuito cabe en un 16V8 de 20 terminales, ¿por qué no especificar aquel que es ligeramente más grande, el 20V8 de 24 terminales de modo que tengamos disponibles entradas de sobra en caso de algún problema? Y, una vez que se haya especificado un 20V8, ¿por qué no hacer uso del 22V10 que tiene algo más de capacidad y que viene en el mismo encapsulado de 24 terminales?

En el mundo real del diseño e ingeniería de productos, la limitante es el costo. De otro modo, el argumento del párrafo anterior podría ser extendido *ad nausem* usando los CPLD y FPGA con todavía más capacidad (vea las secciones 10.5 y 10.6).

Como los automóviles caros y los buenos vinos, los dispositivos digitales tales como los PLD, los CPLD y los FPGA no siempre están cotizados de manera proporcional a sus capacidades y beneficios. En particular, entre más cerca se encuentre la capacidad del dispositivo al "borde del rendimiento posible", mayor será el recargo que puede esperar pagar. De esta forma, cuando se seleccionan dispositivos para realizar un diseño, se deben evaluar muchos factores de peso. Por ejemplo, un CPLD o FPGA de alta densidad muy costoso puede permitir que un diseño sea realizado en un solo dispositivo cuyas funciones internas se modifiquen fácilmente si se necesita. Por otra parte, al utilizar dos o más PLD, CPLD o FPGA de menor densidad puede ahorrarnos el costo del componente pero aumentar el área de la placa y el consumo de energía, al mismo tiempo que se hace más difícil de cambiar el diseño posteriormente (puesto que las interconexiones del dispositivo deben fijarse cuando la placa se fabrique).

Lo que esto quiere demostrar es que el costo total siempre debe considerarse junto con la elegancia y conveniencia del diseño para crear un producto que tenga éxito (es decir, que sea rentable). Y minimizar el costo de un producto por lo regular involucra una plétora de sentido común en la economía y consideraciones de diseño que tienen muy poco que ver con los métodos de minimización de compuerta algorítmicos del capítulo 4.

8.4 Contadores

contador
módulo

Contador módulo m
contador de división entre m

El nombre *contador* generalmente se utiliza para cualquier circuito secuencial temporizado cuyo diagrama de estado contenga un solo lazo, como en la figura 8-26. El *módulo* de un contador es el número de estados en el lazo. Un contador con m estados se conoce como un *contador de módulo m*, o a veces, un *contador de división entre m*. Un contador con un módulo distinto de potencia de 2 tiene estados extras que no se utilizan en la operación normal.

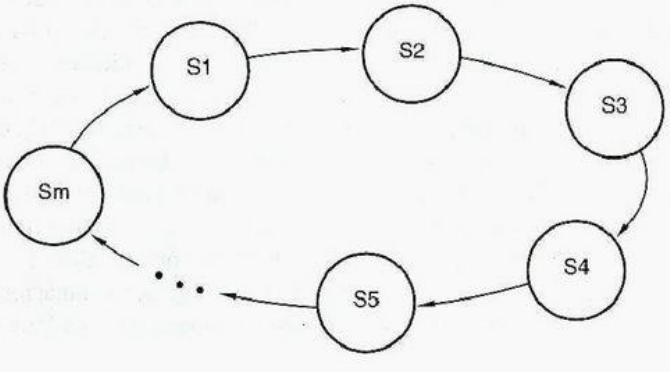


Figura 8-26
Estructura general de un diagrama de estado del contador: un solo lazo.

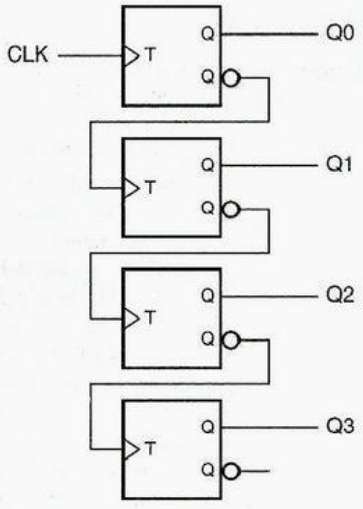


Figura 8-27
Un contador iterativo binario de 4 bits.

contador binario de n bits

Probablemente el tipo de contador que se utiliza con mayor frecuencia es un *contador binario de n bits*. Un contador de esta clase tiene n flip-flops (dispositivos biestables) y 2^n estados, que son visitados en la secuencia 0, 1, 2, ..., 2^n-1, 0, 1, ... Cada uno de estos estados se encuentra codificado como el correspondiente entero binario de n bits.

8.4.1 Contadores iterativos

Un contador binario de n bits puede construirse con solamente n flip-flops y ningún otro componente, para cualquier valor de n. La figura 8-27 muestra un contador de esta clase para $n = 4$. Recordemos que un flip-flop tipo T cambia de estado (conmuta) en cada flanco ascendente de su entrada de reloj. Así, cada bit del contador conmuta si y sólo si el bit inmediatamente anterior cambia de 1 a 0. Esto corresponde a una secuencia de conteo binaria normal, cuando un bit particular cambia de 1 a 0, genera un acarreo al siguiente bit más significativo. El contador se denomina *contador iterativo* porque la información acarreada se propaga ("iterativos") desde los bits menos significativos hasta los bits más significativos, un bit a la vez.

contador iterativo

8.4.2 Contadores sincrónicos

Aunque un contador iterativo requiere menos componentes que cualquier otro tipo de contador binario, esto tiene un precio: es más lento que cualquier otro tipo de contador binario. En el peor de los casos, cuando el bit más significativo debe cambiar, la salida no es válida sino hasta el tiempo $n \cdot t_{TQ}$ después del flanco ascendente de CLK, donde t_{TQ} es el retardo de propagación desde la entrada hasta la salida de un flip-flop T.

contador sincrónico

Un *contador sincronico* conecta la totalidad de las entradas de reloj de su flip-flop a la misma señal CLK común, de modo que todas las salidas del flip-flop cambien al mismo tiempo, sólo después t_{TQ} ns de retardo. Como se muestra en la figura 8-28, esto requiere del uso de flip-flops T con entradas de habilitación; la salida se conmuta en el flanco ascendente de T si y sólo si EN está asertiva. La lógica combinacional en las entradas EN determina cuál, si existe, de los flip-flops se conmuta en cada flanco ascendente de T.

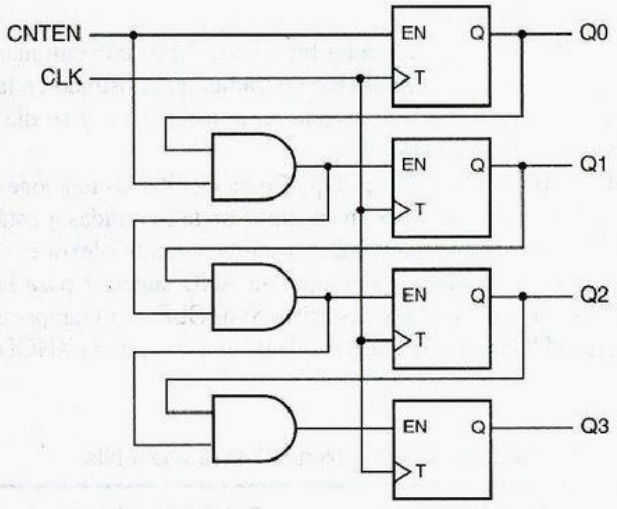


Figura 8-28
Un contador binario sincrónico de 4 bits con lógica de habilitación serie.

Como se ilustra en la figura 8-28, también es posible proporcionar una señal de habilitación de conteo maestra CNTEN. Cada flip-flop T se conmuta si y sólo si CNTEN es asertiva y todos los bits del contador de orden inferior son 1. Como el contador iterativo binario, un contador binario sincrónico de n bits puede construirse con una cantidad fija de lógica por bit; en este caso, un flip-flop T con habilitación y una compuerta AND de 2 entradas.

contador sincrónico serie

La estructura del contador en la figura 8-28 en ocasiones se denomina un *contador sincrónico serie* porque las señales de habilitación combinacionales se propagan en serie desde los bits menos significativos hasta los más significativos. Si el periodo del reloj es demasiado corto, puede no haber suficiente tiempo para que un cambio en el LSB del contador se propague al MSB. Este problema se elimina en la figura 8-29 al controlar cada entrada EN con una compuerta AND dedicada, con sólo un nivel de lógica. Ésta es la estructura de contador binario más rápida, llamada *contador sincrónico en paralelo.*

contador sincrónico en paralelo

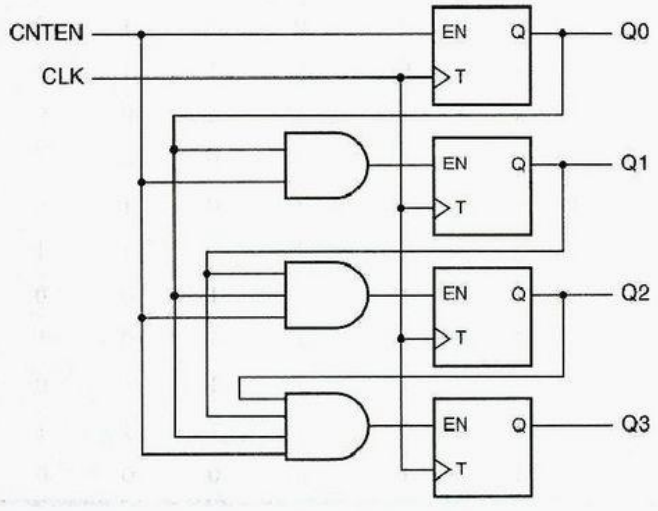


Figura 8-29
Un contador binario sincrónico de 4 bits con lógica de habilitación paralela

8.4.3 Contadores MSI y aplicaciones

74x163

El más popular contador MSI es el *74x163*, un contador binario de 4 bits con entradas de carga y borrado de nivel activo bajo con el símbolo lógico tradicional ilustrado en la figura 8-30. Su función es resumida por la tabla de estado en la tabla 8-11, y su diagrama lógico interno se muestra en la figura 8-31.

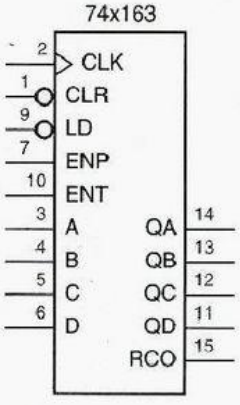


Figura 8-30 Símbolo lógico tradicional para la 74x163.

El '163 utiliza flip-flops D internos en vez de flip-flops T para facilitar las funciones de carga y borrado. Cada entrada D es controlada por un multiplexor de 2 entradas, y está formada por una compuerta OR y dos compuertas AND. La salida del multiplexor es 0 si la entrada CLR_L es asertiva. De otro modo, la compuerta AND superior pasa la entrada de datos (A, B, C o D) a la salida si LD_L es asertiva. Si ni CLR_L ni tampoco LD_L son asertivas, la compuerta AND inferior pasa la salida de una compuerta XNOR a la salida del multiplexor.

Tabla 8-11 Tabla de estado para un contador binario 74x163 de 4 bits.

Entradas				*Estado actual*				*Estado siguiente*			
CLR_L	*LD_L*	*ENT*	*ENP*	*QD*	*QC*	*QB*	*QA*	*QD**	*QC**	*QB**	*QA**
0	x	x	x	x	x	x	x	0	0	0	0
1	0	x	x	x	x	x	x	D	C	B	A
1	1	0	x	x	x	x	x	QD	QC	QB	QA
1	1	x	0	x	x	x	x	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

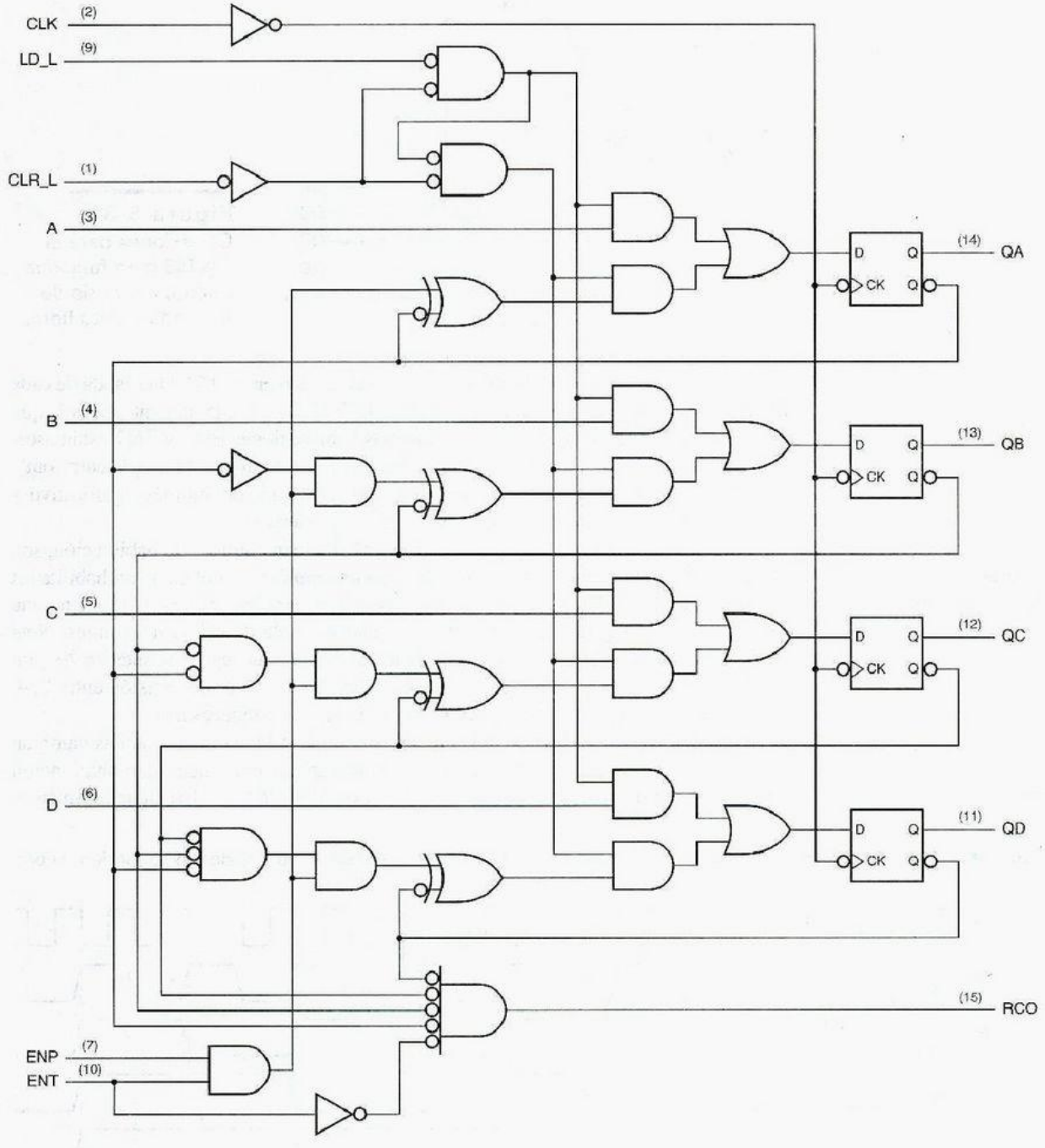


Figura 8-31 Diagrama lógico para el contador binario sincrónico de 4 bits 74x163, incluyendo números de terminales para un encapsulado en línea doble estándar de 16 terminales.

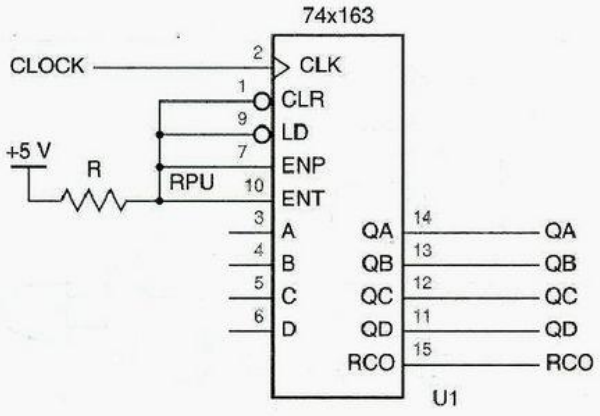


Figura 8-32
Conexiones para el 74x163 para funcionamiento en modo de funcionamiento libre.

Las compuertas XNOR efectúan la función de conteo en el '163. Una salida de cada XNOR es el correspondiente bit de conteo (QA, QB, QC o QD); la otra entrada es 1, que complementa el bit de conteo, si y sólo si ambas habilitaciones ENP y ENT están asertivas y todos los bits de conteo de menor orden son 1. La señal RCO ("ripple carry out", acarreo de salida iterativo) indica un acarreo desde la posición del bit más significativo y es 1 cuando todos los bits de conteo son 1 y ENT es asertiva.

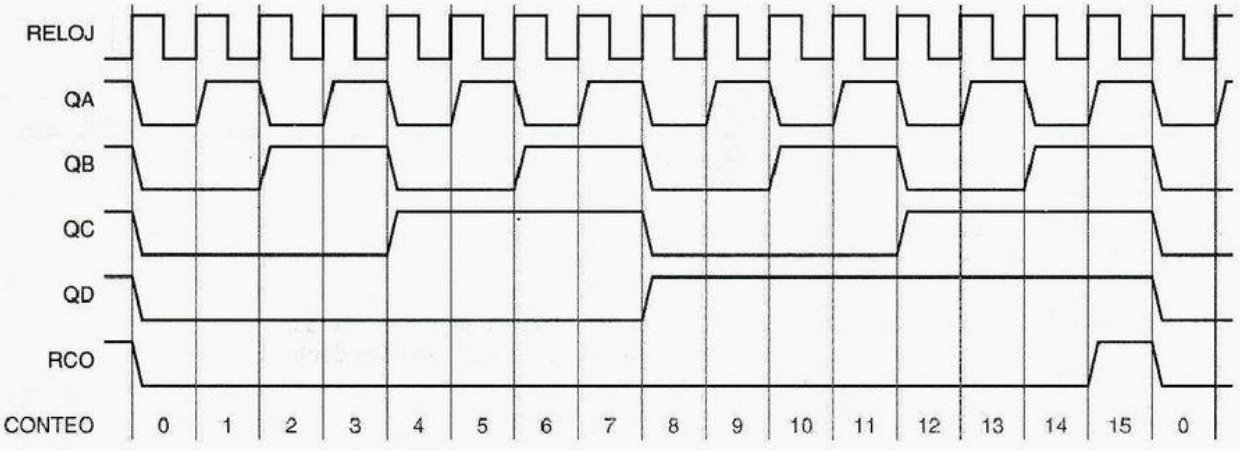
contador de funcionamiento libre

Aun cuando la mayoría de los contadores MSI tienen entradas de habilitación, son utilizadas con frecuencia en un modo de *funcionamiento libre* en el cual son habilitadas continuamente. La figura 8-32 muestra las conexiones para hacer que un '163 funcione en este modo y la figura 8-33 muestra las formas de onda de salida resultantes. Note que, comenzando con QA, cada señal tiene la mitad de la frecuencia de la anterior. Así, un '163 de funcionamiento libre puede utilizarse como un contador de división entre 2, 4, 8 o 16, ignorando cualquier bit de salida de orden superior innecesario.

74x161

Advierta que el '163 es completamente sincrónico; es decir, sus salidas cambian solamente en el flanco ascendente de CLK. Algunas aplicaciones necesitan una función de borrado asincrónico, como la proporcionada por el *74x161*. El '161 tiene las mismas

Figura 8-33 Formas de onda de reloj y de salida para un contador de división entre 16 de funcionamiento libre.



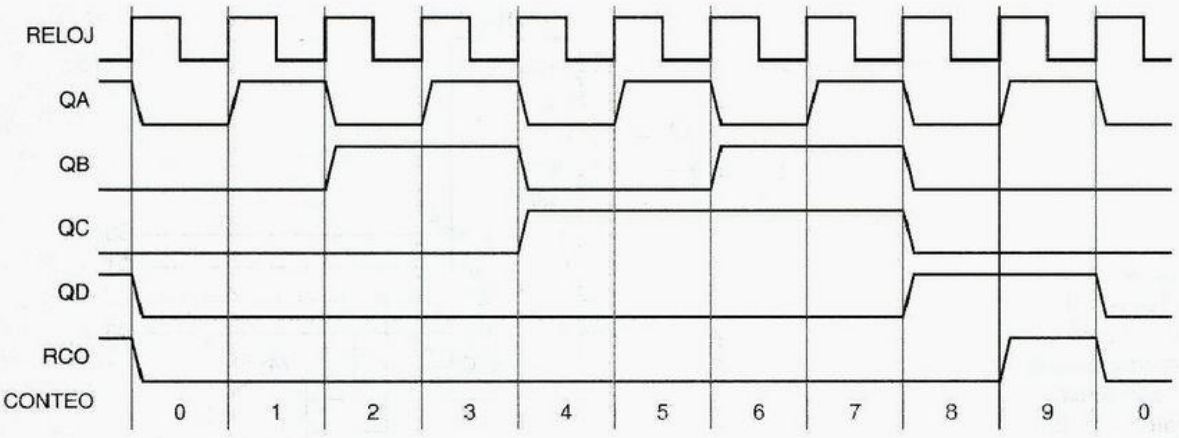


Figura 8-34 Formas de onda de reloj y de salida para un contador de división entre 10 de funcionamiento libre.

terminales de salida del '163, pero su entrada CLR_L está conectada a las entradas de borrado asincrónicas de sus flip-flops.

El *74x160* y el *74x162* son más variaciones con las mismas terminales de salida y funciones generales que el '161 y el '163, excepto que la secuencia de conteo es modificada para ir al estado 0 después del estado 9. En otras palabras, son contadores módulo 10, llamados a veces *contadores de décadas*. La figura 8-34 muestra las formas de onda de salida para un '160 o '162 de funcionamiento libre. Note que aunque las salidas QD y QC tienen un décimo de la frecuencia de CLK no tienen un lazo de trabajo del 50%, y la salida QC, con una quinta parte de la frecuencia de entrada, no tiene un lazo de trabajo constante. Mostraremos el diseño de un contador con división entre 10 con un 50% de salida de lazo de trabajo posteriormente en esta subsección.

74x160
74x162

contador de décadas

Aunque el '163 es un contador de módulo 16, puede hacerse que cuente en un módulo menor de 16 utilizando la entrada CLR_L o LD_L para abreviar la secuencia normal de conteo. Por ejemplo, la figura 8-35 muestra una manera de emplear el '163 como un contador módulo 11. La salida RCO, que detecta el estado 15, se utiliza para forzar el estado

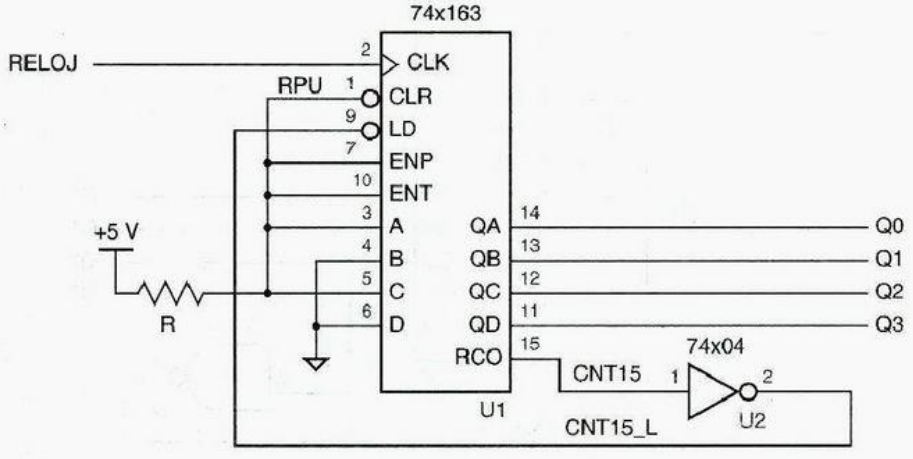


Figura 8-35
Uso del 74x163 como un contador módulo 11 con la secuencia de conteo 5, 6, ..., 15, 5, 6, ...

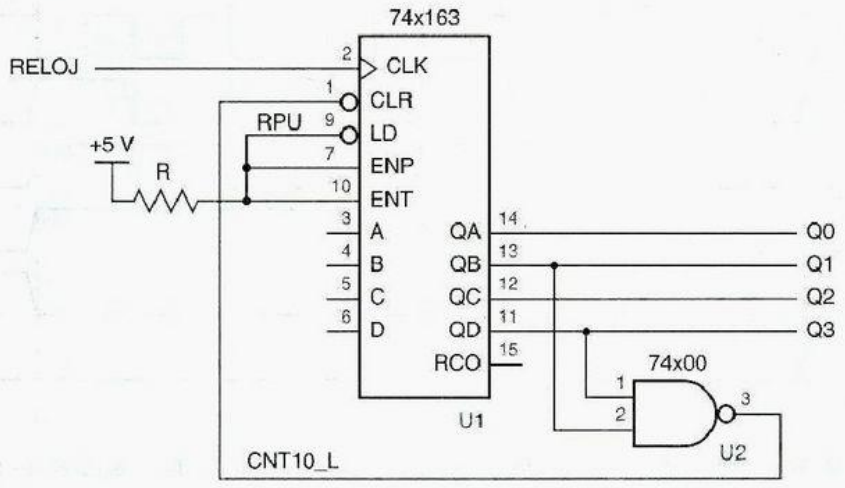


Figura 8-36
Uso del 74x163 como un contador módulo 11 con la secuencia de conteo 0, 1, 2, ..., 10, 0, 1, ...

siguiente a 5, de modo que el circuito contará de 5 a 15 y después comenzará en 5 de nuevo, para un total de 11 estados por lazo de conteo.

Un enfoque diferente para el conteo de módulo 11 con el '163 es mostrado en la figura 8-36. Este circuito utiliza una compuerta NAND para detectar el estado 10 y forzar el estado siguiente a 0. Note que solamente una compuerta de 2 entradas se emplea para detectar el estado 10 (binario 1010). Aunque una compuerta de 4 entradas se usaría normalmente para detectar la condición $CNT10 = Q3 \cdot Q2' \cdot Q1 \cdot Q0'$, la compuerta de 2 entradas se aprovecha del hecho de que ningún otro estado en la secuencia normal de conteo de 0–10 tiene $Q3 = 1$ y $Q1 = 1$. En general, para detectar el estado N en un contador binario que cuenta de 0 a N, necesitamos aplicar AND solamente a los bits de estado que son 1 en la codificación binaria de N.

Existen muchas otras maneras de hacer un contador módulo 11 con la ayuda de un '163. La elección del enfoque (uno de los anteriores o una combinación de ellos, como en el ejercicio 8.36) depende de la aplicación. Como otro ejemplo, en la sección 2.10 prometimos mostrarle a usted cómo construir un circuito que cuente en el código decimal de

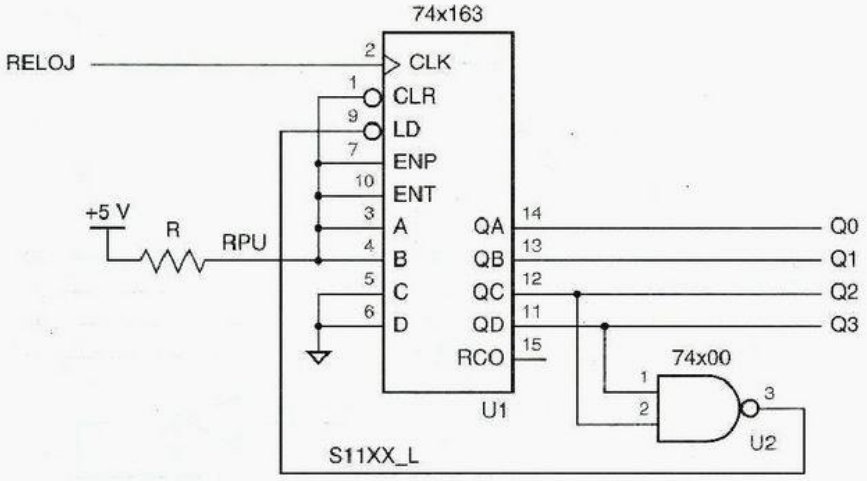


Figura 8-37
Un 74x163 que se utiliza como un contador decimal de exceso 3.

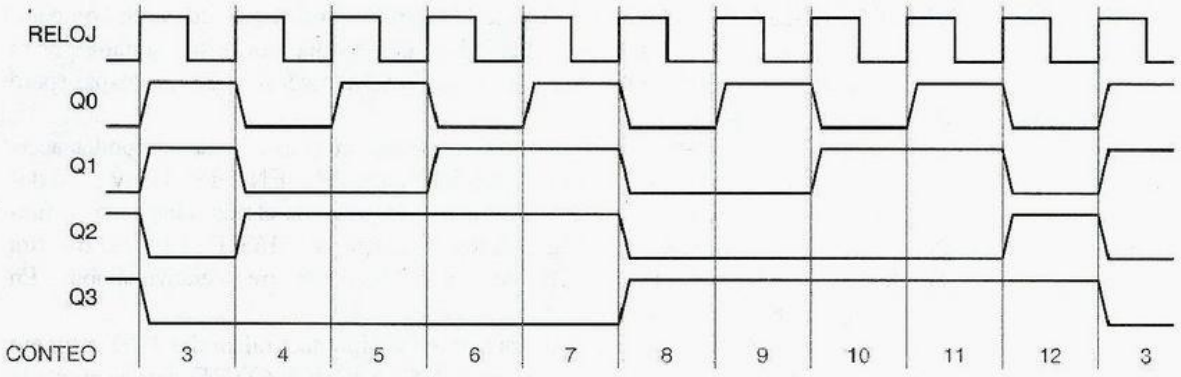
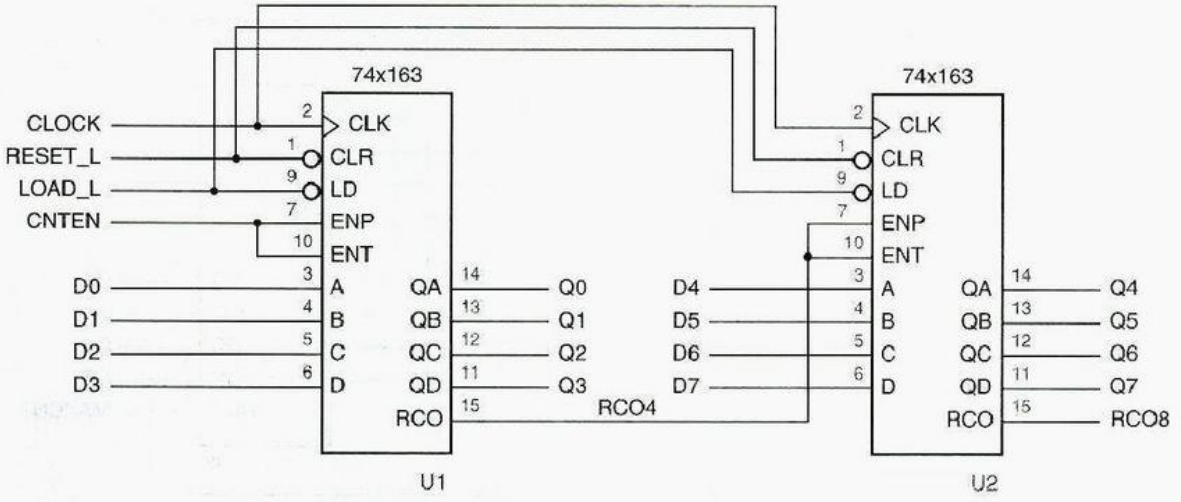


Figura 8-38 Formas de onda para el '163 que se utiliza como un contador decimal de exceso 3.

exceso 3, mostrado en la tabla 2-9. La figura 8-37 muestra las conexiones para que un '163 cuente en secuencia de exceso 3. Una compuerta NAND detecta el estado 1100 y obliga a que 0011 sea cargado como el estado siguiente. La figura 8-38 muestra las formas de onda de tiempo resultantes. Advierta que la salida Q3 tiene 50% de ciclo de trabajo, lo que puede ser deseable para algunas aplicaciones.

Un contador binario con un módulo mayor que 16 puede construirse al poner en cascada 74x163s. La figura 8-39 muestra las conexiones generales para un contador de esta clase. Las entradas CLK, CLR_L y LD_L de todos los '163 se conectan en paralelo, de modo que todos cuentan, con borrados o cargados al mismo tiempo. Una señal maestra de habilitación de conteo (CNTEN) se conecta al '163 de menor orden. La salida RCO4 es asertiva si y sólo si el '163 de menor orden se encuentra en estado 15 y CNTEN es asertiva; RCO4 se conecta a las entradas de habilitación del '163 de orden mayor. Así, tanto la información acarreada como la habilitación maestra de conteo se propagan desde la salida de una etapa del contador de 4 bits a la siguiente. Como el contador sincrónico serie

Figura 8-39 Conexiones generales en cascada para contadores basados en el 74x163.



de la figura 8-28, este esquema puede extenderse para construir un contador con cualquier número de bits deseado; la velocidad de conteo máximo está limitada por el retardo de propagación de la señal de acarreo iterativo a través de todas las etapas (pero vea el ejercicio 8.38).

Incluso los diseñadores digitales experimentados en ocasiones se confunden acerca de la diferencia entre las entradas de habilitación ENP y ENT del '163 y contadores similares, puesto que ambas deben ser asertivas para que el contador haga su función. Sin embargo, un vistazo al diagrama lógico interno del '163, figura 8-31, muestra la diferencia con bastante claridad: ENT se va a la salida de acarreo iterativo también. En muchas aplicaciones esta distinción es importante.

Por ejemplo, la figura 8-40 muestra una aplicación que utiliza dos '163 como un contador módulo 193 que cuenta desde 63 hasta 255. La salida MAXCNT detecta el estado 255 y detiene el contador hasta que GO_L es asertiva. Cuando esto ocurre, el contador se vuelve a cargar con 63 y de nuevo cuenta hasta 255. (Note que el valor de GO_L es relevante solamente cuando el contador se encuentra en el estado 255). Para mantener el contador detenido, MAXCNT debe ser asertivo en el estado 255 incluso mientras el con-

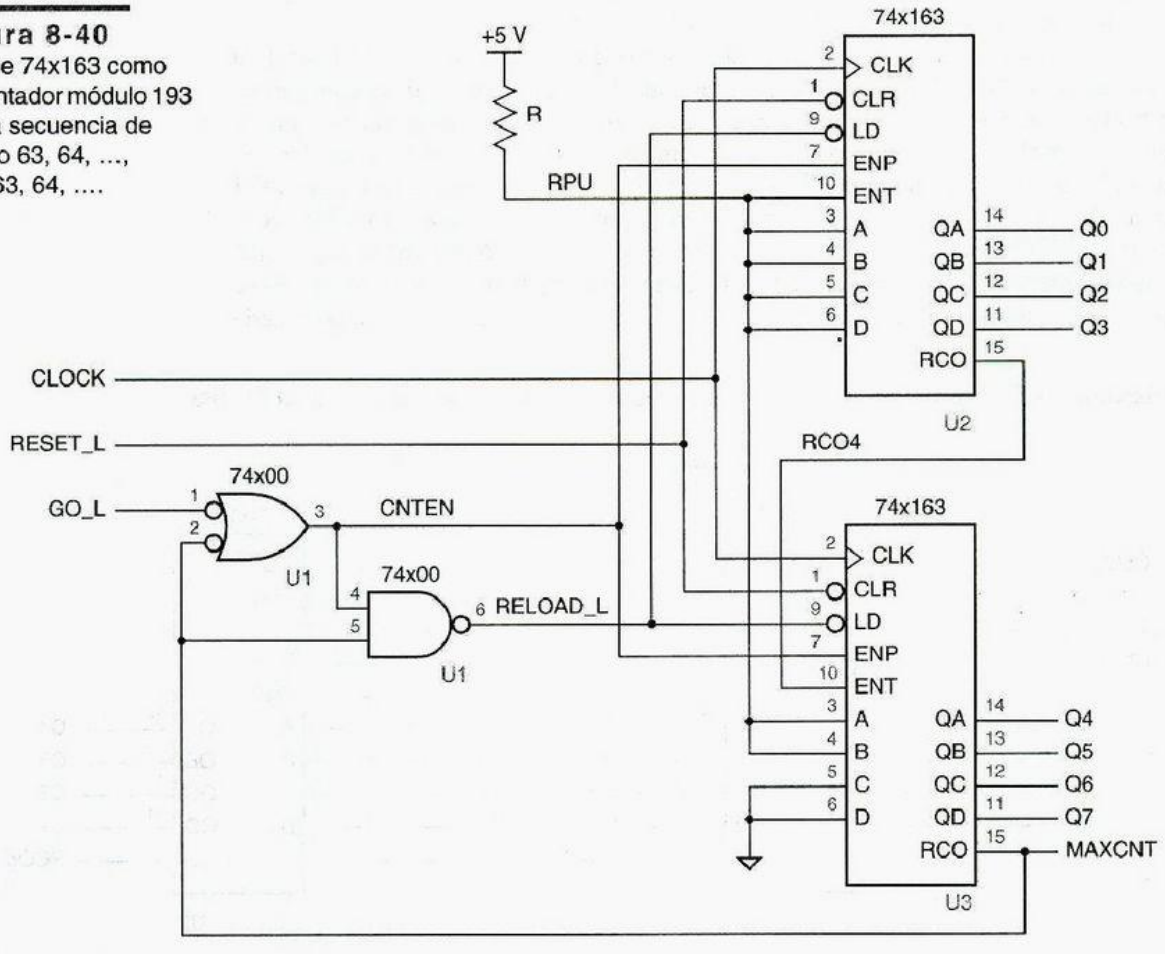


Figura 8-40
Uso de 74x163 como un contador módulo 193 con la secuencia de conteo 63, 64, ..., 255, 63, 64, ...

tador se encuentre detenido. Por tanto, la entrada ENT del contador de orden inferior siempre se encuentra asertiva, y la salida RCO se encuentra conectada a la entrada ENT de orden superior y MAXCNT detecta el estado 255 incluso si CNTEN no está asertiva (compárese con el comportamiento de RCO8 en la figura 8-39). Para habilitar el conteo, CNTEN se conecta a las entradas ENP en paralelo. Una compuerta NAND afirma RELOAD_L para regresar al estado 63 solamente si GO_L es asertiva y el contador se encuentra en el estado 255.

74x169

Otro contador con funciones similares a las del 74x163 es el *74x169*, cuyo símbolo lógico se muestra en la figura 8-41. Una diferencia en el '169 es que su salida de acarreo y sus entradas de habilitación son de nivel activo bajo. Más importante, el '169 es un *contador arriba/abajo*; cuenta en orden binario ascendente o descendente dependiendo del valor de una señal de entrada, UP/DN. El '169 hacia arriba cuando UP/DN es 1 y hacia abajo cuando UP/DN es 0.

Contador ascendente/ descendente

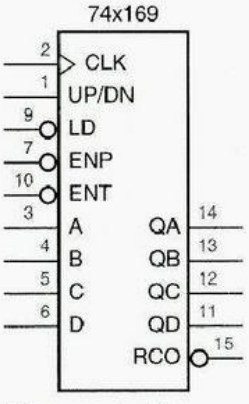


Figura 8-41
Símbolo lógico para el contador ascendente/ descendente 74x169.

8.4.4 Decodificación de estados de contador binario

Un contador binario puede ser combinado con un decodificador para obtener un conjunto de señales codificadas 1 de *m*, donde una señal es asertiva en cada estado del contador. Esto es útil cuando los contadores se emplean para controlar un conjunto de dispositivos, donde un dispositivo diferente es habilitado en cada estado del contador. En este enfoque, cada salida del decodificador habilita un dispositivo diferente.

La figura 8-42 muestra cómo un 74x163 alambrado como un contador módulo 8 puede ser combinado con un decodificador de 3 a 8 74x138 para proporcionar ocho señales, cada una representando un estado del contador. La figura 8-43 muestra la temporización típica para este circuito. Cada salida del decodificador es asertiva durante un periodo de reloj correspondiente

Note que las salidas del decodificador pueden contener "interferencias" ("glitches") en transiciones de estado donde dos o más bits de contador cambian, aun cuando las salidas del '163 estén libres de interferencias y el '138 no tenga ningún riesgo estático. En un contador sincrónico como el '163, las salidas no cambian exactamente al mismo tiempo. Más importante aún, es el hecho de que las trayectorias de señal múltiple en un

Interferencias de decodificación

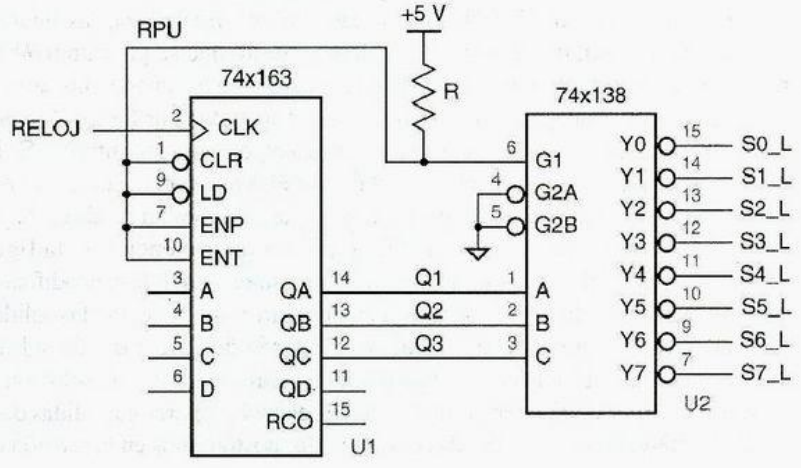


Figura 8-42
Un decodificador y contador binario módulo 8.

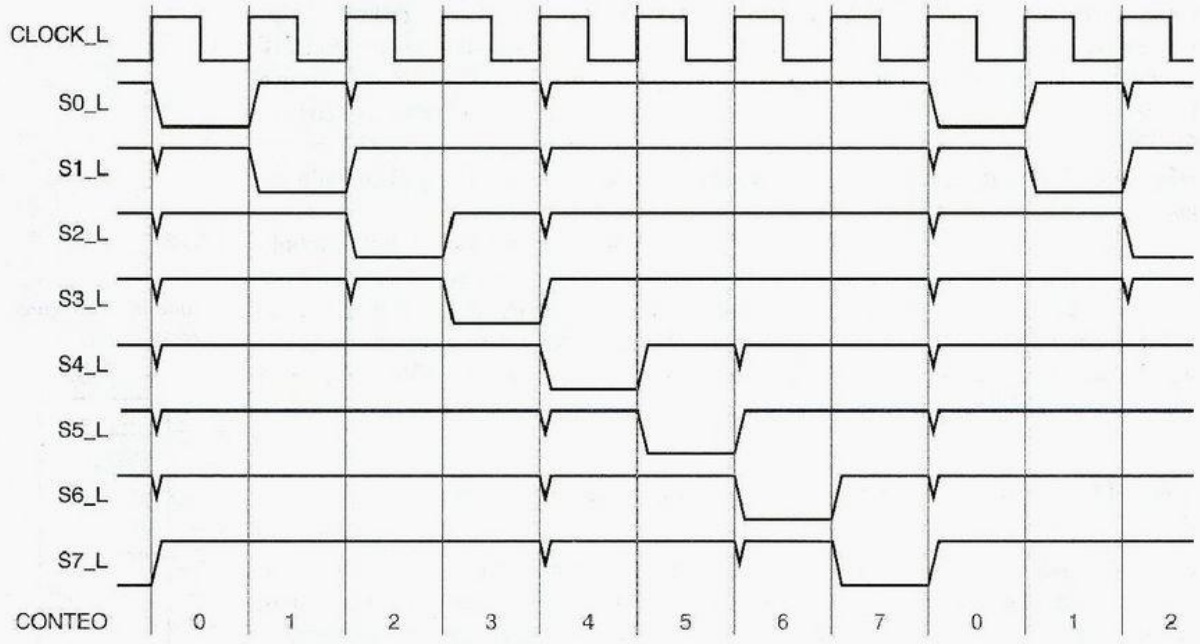


Figura 8-43 Diagrama de temporización para un contador y decodificador binario módulo 8, mostrando las interferencias de decodificación.

decodificador como el '138 tienen diferentes retardos; por ejemplo, la trayectoria desde B hasta Y1_L es más rápida que la trayectoria desde A hasta Y1_L. De este modo, aun cuando la entrada cambia simultáneamente de 011 a 100, el decodificador puede comportarse como si la entrada fuera temporalmente 001, y la salida Y1_L puede tener una interferencia. En el ejemplo presente, puede mostrarse que las interferencias pueden ocurrir *en cualquier* realización de la función del decodificador binario; este problema es un ejemplo de un *riesgo de función*.

riesgo de función

En la mayoría de las aplicaciones, las señales de salida del decodificador representadas en la figura 8-43 se utilizarían como entradas de control para registros, contadores, y otros dispositivos disparados por flanco (por ejemplo, EN_L en un 74x377, LD_L en un 74x163 o ENP_L en un 74x169). En un caso de esta naturaleza, las interferencias de decodificación en la figura no son un problema, puesto que se presentan *después* del tic de reloj. Se han marchado antes de que llegue el siguiente tic, cuando las salidas del decodificador son muestreadas por otros dispositivos disparados por flanco. Sin embargo, las interferencias *serían* un problema si se aplicaran a algo como las entradas S_L o R_L de un cerrojo $\overline{S}$-$\overline{R}$. De igual forma, el uso de tales señales que potencialmente tienen interferencias como relojes para dispositivos disparados por flanco es un no absoluto y definitivo.

Si es necesario, una manera de "limpiar" las interferencias en la figura 8-43 es conectar las salidas del '138 a otro registro que muestree las salidas decodificadas estables en el siguiente tic de reloj, como se ilustra en la figura 8-44. Note que las salidas decodificadas han sido renombradas para explicar el retardo de 1 tic a través del registro. No obstante, una vez que usted decide pagar por un registro de 8 bits, una solución menos costosa es emplear un "contador en anillo" de 8 bits, el cual proporciona salidas decodificadas libres de interferencias de manera directa, como lo mostraremos en la sección 8.5.6.

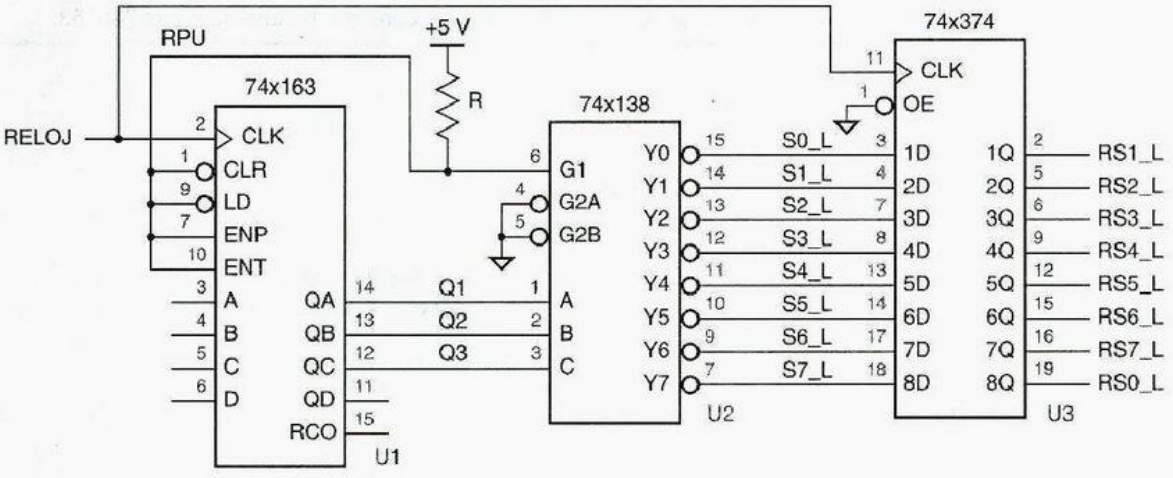


Figura 8-44 Un contador y decodificador binario módulo 8 con salidas libres de interferencias.

8.4.5 Contadores en ABEL y VHDL

Los contadores binarios son buenos candidatos para diseño basado en ABEL y PLD, por varias razones:

- Una máquina de estado grande puede con frecuencia ser descompuesta en dos o más pequeñas máquinas de estado, donde una de las máquinas más pequeñas es un contador binario que se mantiene al tanto de cuánto tiempo debería permanecer la otra máquina en un estado particular. Esto puede simplificar tanto el diseño conceptual como el diseño del circuito de la máquina.
- Muchas aplicaciones requieren contadores de módulo casi binario con requerimientos especiales para inicialización, detección de estado o salto de estado. Por ejemplo, un contador en un controlador de elevador puede saltar el estado 13. En lugar de emplear un contador binario comercial y lógica extra para los requerimientos especiales, un diseñador puede especificar exactamente las funciones requeridas en un programa ABEL.
- La mayoría de los contadores estándar MSI tienen solamente 4 bits, mientras que un solo PLD de 24 terminales se puede utilizar para crear un contador binario con hasta 10 bits.

El contador MSI más popular es el contador binario de 4 bits 74x163, ilustrado en la figura 8-31. Después de un vistazo a esta figura percibimos que la lógica de excitación para este contador no es exactamente simple, especialmente considerando el uso de las compuertas XNOR. No obstante, ABEL proporciona una manera muy simple de definir el comportamiento del contador, mismo que describiremos más adelante.

Recuerde que ABEL utiliza el símbolo "+" para especificar la suma de enteros. Cuando dos conjuntos son "sumados" con este operador, cada uno es interpretado como un número binario; el elemento más a la derecha del conjunto corresponde al bit menos significativo del número. De este modo, la función de un 74x163 puede especificarse mediante el programa ABEL en la tabla 8-12. Cuando el contador está habilitado, se agrega un 1 al estado actual.

Tabla 8-12 El programa ABEL para un contador binario de 4 bits 74x163.

```
module Z74X163
title '4-bit Binary Counter'

" Input pins
CLK, LD_L, CLR_L, ENP, ENT      pin;
A, B, C, D                      pin;
" Output pins
QA, QB, QC, QD                  pin istype 'reg';
RCO                             pin istype 'com';

" Set definitions
INPUT = [D, C, B, A];
COUNT = [QD, QC, QB, QA];

equations

COUNT.CLK = CLK;

COUNT := !CLR & ( LD & INPUT
                # !LD & (ENT & ENP) & (COUNT + 1)
                # !LD & !(ENT & ENP) & COUNT);

RCO = (COUNT == [1,1,1,1]) & ENT;

end Z74X163
```

Tabla 8-13 Ecuaciones minimizadas para el contador binario de 4 bits de la tabla 8-12.

```
QA := (CLR_L & LD_L & ENT & ENP & !QA
     # CLR_L & LD_L & !ENP & QA
     # CLR_L & LD_L & !ENT & QA
     # CLR_L & !LD_L & A);

QB := (CLR_L & LD_L & ENT & ENP & !QB & QA
     # CLR_L & LD_L & QB & !QA
     # CLR_L & LD_L & !ENP & QB
     # CLR_L & LD_L & !ENT & QB
     # CLR_L & !LD_L & B);

QC := (CLR_L & LD_L & ENT & ENP & !QC & QB & QA
     # CLR_L & LD_L & QC & !QA
     # CLR_L & LD_L & QC & !QB
     # CLR_L & LD_L & !ENP & QC
     # CLR_L & LD_L & !ENT & QC
     # CLR_L & !LD_L & C);

QD := (CLR_L & LD_L & ENT & ENP
            & !QD & QC & QB & QA
     # CLR_L & !LD_L & D
     # CLR_L & LD_L & QD & !QB
     # CLR_L & LD_L & QD & !QC
     # CLR_L & LD_L & !ENP & QD
     # CLR_L & LD_L & !ENT & QD
     # CLR_L & LD_L & QD & !QA);

RCO = (ENT & QD & QC & QB & QA);
```

La tabla 8-13 muestra las ecuaciones lógicas minimizadas que genera ABEL para el contador de 4 bits. Note que cada bit de salida significativo requiere de un término de producto adicional. Como resultado, el tamaño de los contadores que pueden realizarse en un 16V8 o incluso un 20V8 está generalmente limitado a cinco o seis bits. Otros dispositivos, incluyendo los PLD serie X y algunos CPLD, contienen una estructura XOR que puede realizar contadores grandes sin incrementar los requerimientos de términos de producto.

El diseño de una secuencia de conteo especializada en ABEL es mucho más simple que adaptar un contador binario estándar. Por ejemplo, el programa ABEL en la tabla 8-12 puede ser adaptado para contar en secuencia de exceso 3 (figura 8-38) cambiando las ecuaciones como se ve a continuación:

```
COUNT := !CLR & ( LD & INPUT
       # !LD & (ENT & ENP) &
           ((COUNT==12) & 3) # ((COUNT!=12) & (COUNT + 1))
       # !LD & !(ENT & ENP) & COUNT);

RCO = (COUNT == 12) & ENT;
```

Los PLD pueden conectarse en cascada para obtener contadores más amplios al proporcionar a cada etapa del contador una salida de acarreo que indique cuando se encuentre cerca de completar una vuelta. Existen dos enfoques básicos para generar la salida de acarreo:

- *Combinacional.* La ecuación de acarreo indica que el contador se encuentra habilitado y está actualmente en su último estado antes de completar la vuelta. Para un contador binario hacia arriba de 5 bits, tenemos

  salida de acarreo combinacional

  ```
  COUT = CNTEN & Q4 & Q3 & Q2 & Q1 & Q0;
  ```

  Puesto que se incluye CNTEN, este enfoque permite que los acarreos sean enlazados a través de contadores en cascada al conectar cada COUT al siguiente CNTEN.

- *Registrado.* La ecuación de acarreo indica que el contador está cerca de entrar a su último estado antes de completar la vuelta. De esta manera, en el siguiente tic de reloj, el contador entra a este último estado y la salida de acarreo es asertiva. Para un contador binario ascendente de 5 bits con entradas de carga y borrado, tenemos

  salida de acarreo registrado

  ```
  COUT := !CLR & !LD & CNTEN
            & Q4 & Q3 & Q2 & Q1 & !Q0
        # !CLR * !LD * !CNTEN
            & Q4 & Q3 & Q2 & Q1 & Q0
        # !CLR & LD
            & D4 & D3 & D2 & D1 & D0;
  ```

El segundo enfoque tiene la ventaja de producir COUT con menos retardo que el enfoque combinacional. Sin embargo, las compuertas externas son requeridas ahora entre etapas, puesto que la señal CNTEN para cada etapa debería ser el AND lógico de la señal maestra de habilitación de conteo y las salidas COUT de todos los contadores de orden inferior. Estas compuertas externas pueden evitarse si los contadores de orden superior tienen múltiples entradas de habilitación.

8.4.6 Contadores en VHDL

Como en ABEL, VHDL permite que los contadores sean especificados con bastante facilidad. El reto más grande en VHDL, con su fuerte verificación de tipos, es conseguir que todos los tipos de señal estén definidos de manera correcta y consistente.

La tabla 8-14 es un programa VHDL para un contador binario tipo 74x163. Note que el programa hace uso de la biblioteca `IEEE.std_logic_arith.all`, la cual incluye el tipo `UNSIGNED`, como lo describimos en la sección 5.9.6 de la página 427. Esta biblioteca incluye definiciones de operadores "+" y "−" que realizan la suma y resta sin signo de operandos `UNSIGNED`. El programa contador declara la entrada y salida del contador como vectores `UNSIGNED` y utiliza "+" para incrementar el valor del contador como sea requerido.

En el programa definimos una señal interna `IQ` para retener el valor del contador. Podríamos haber empleado `Q` directamente, pero entonces tendríamos que declarar su tipo de puerto como `buffer` en vez de `out`. También, podríamos haber definido el tipo de puertos `D` y `Q` para ser `STD_LOGIC_VECTOR`, pero entonces tendríamos que efectuar conversiones de tipo dentro del cuerpo del proceso (vea el ejercicio 8-51).

Tabla 8-14 Programa VHDL para un contador binario de 4 bits tipo 74x163.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity V74x163 is
    port ( CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
           D: in UNSIGNED (3 downto 0);
           Q: out UNSIGNED (3 downto 0);
           RCO: out STD_LOGIC );
end V74x163;

architecture V74x163_arch of V74x163 is
signal IQ: UNSIGNED (3 downto 0);
begin
process (CLK, ENT, IQ)
  begin
    if (CLK'event and CLK='1') then
      if CLR_L='0' then IQ <= (others => '0');
      elsif LD_L='0' then IQ <= D;
      elsif (ENT and ENP)='1' then IQ <= IQ + 1;
      end if;
    end if;
    if (IQ=15) and (ENT='1') then RCO <= '1';
    else RCO <= '0';
    end if;
    Q <= IQ;
  end process;
end V74x163_arch;
```

Tabla 8-15 Arquitectura VHDL para conteo en orden de exceso 3.

```
architecture V74xs3_arch of V74x163 is
signal IQ: UNSIGNED (3 downto 0);
begin
process (CLK, ENT, IQ)
  begin
    if CLK'event and CLK='1' then
      if CLR_L='0' then IQ <= (others => '0');
      elsif LD_L='0' then IQ <= D;
      elsif (ENT and ENP)='1' and (IQ=12) then IQ <= ('0','0','1','1');
      elsif (ENT and ENP)='1' then IQ <= IQ + 1;
      end if;
    end if;
    if (IQ=12) and (ENT='1') then RCO <= '1';
    else RCO <= '0';
    end if;
    Q <= IQ;
  end process;
end V74xs3_arch;
```

Como en ABEL, las secuencias de conteo especializadas se pueden especificar fácilmente utilizando código VHDL de comportamiento. Por ejemplo, la tabla 8-15 modifica el contador tipo 74x163 para que cuente en secuencia de exceso 3 (3, ... , 12, 3, ...).

Desgraciadamente, algunos motores de síntesis VHDL no sintetizan contadores en forma satisfactoria. En particular, tienden a sintetizar el paso de conteo con la ayuda de un sumador binario con el valor del contador y una constante 1 como operandos. Este método requiere de mucho más lógica combinacional que la que hemos mostrado para contadores discretos; es particularmente costosa en CPLD y en los FPGA que contengan flip-flops T, compuertas XOR u otras estructuras optimizadas para contadores. En este caso, una alternativa útil es escribir en VHDL estructural orientado a las celdas disponibles en una tecnología en particular como CPLD, FPGA o ASIC.

Por ejemplo, podemos construir una celda de bit para un contador tipo 74x163 utilizando el circuito en la figura 8-45. Este circuito está diseñado para emplear propagación serie para los bits de acarreo, de modo que el mismo circuito se puede utilizar en cualquier etapa de un contador arbitrariamente grande, sujeto a las restricciones que impone el fan out en las señales comunes que controla todas las etapas. Las señales en la celda de bit tienen las definiciones siguientes:

CLK (común) La entrada de reloj para todas las etapas.

LDNOCLR (común) Es asertiva si la entrada LD del contador es asertiva y CLR es negada.

NOCLRORLD (común) Es asertiva si las entradas CLR y LD del contador son ambas negadas.

CNTENP (común) Es asertiva si la entrada ENP del contador es también asertiva.

Di (por celda) La entrada de datos de carga para la celda *i*.

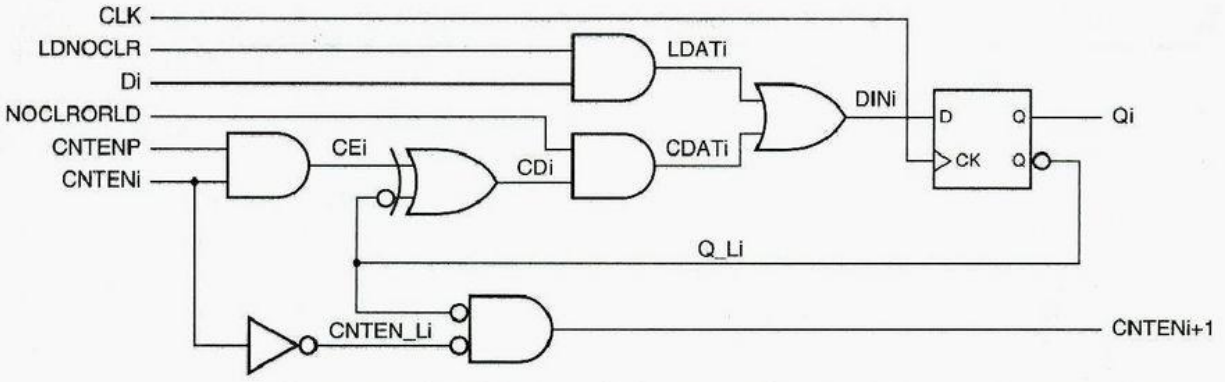


Figura 8-45 Celda de un bit de un contador sincrónico en serie, como el 74x163.

CNTENi (por celda) Entrada de habilitación de conteo serie para la celda *i*.

CNTENi+1 (por celda) Salida de habilitación de conteo serie para la celda *i*.

Qi (por celda) La salida del contador para la celda *i*.

La tabla 8-16 es un programa VHDL correspondiente a la celda de bit en la figura. En el programa, se supone ya definido el componente de flip-flop D `Vdffqqn`; es semejante al flip-flop D en la tabla 8-6 con la adición de una salida QN (complementada). En un diseño FPGA o ASIC, un tipo de componente flip-flop sería elegido de la biblioteca de celda estándar del fabricante.

Tabla 8-16 Programa VHDL para celda de contador de la figura 8-45.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity syncsercell is
  port( CLK, LDNOCLR, NOCLRORLD, CNTENP, D, CNTEN: in STD_LOGIC;
        CNTENO, Q: out STD_LOGIC );
end syncsercell;

architecture syncsercell_arch of syncsercell is
component Vdffqqn
  port( CLK, D: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end component;
signal LDAT, CDAT, DIN, Q_L: STD_LOGIC;
begin
  LDAT <= LDNOCLR and D;
  CDAT <= NOCLRORLD and ((CNTENP and CNTEN) xor not Q_L);
  DIN  <= LDAT or CDAT;
  CNTENO <= (not Q_L) and CNTEN;
  U1: Vdffqqn port map (CLK, DIN, Q, Q_L);
end syncsercell_arch;
```

CUESTIÓN DE ESTILO Advierta que la tabla 8-16 hace uso de una combinación de estilos VHDL de flujo de datos y estructural. Podría haberse escrito completamente estructural, utilizando por ejemplo las definiciones de componente del fabricante ASIC, para garantizar que el circuito sintetizado concuerda exactamente con la figura 8-45. Sin embargo, la mayoría de los motores de síntesis pueden hacer un buen trabajo de selección de la mejor realización de compuerta para las asignaciones de señal empleadas aquí.

La tabla 8-17 muestra cómo crear un contador serie sincrónico de 8 bits con la ayuda de la celda que definimos anteriormente. Las primeras dos asignaciones en el cuerpo de la arquitectura sintetizan las señales LDNCLR y NOCLRORLD en común. Las siguientes dos proposiciones manejan las condiciones de frontera para la cadena serie de habilitación de conteo. Finalmente, la instrucción `generate` (presentada en la página 453) instancia ocho celdas de contador de 1 bit y conecta la cadena de habilitación de conteo como se requiere.

Debe quedar claro que un contador más grande o más pequeño puede crearse simplemente al cambiar unas cuantas definiciones en el programa. Se puede poner la instrucción `generic` de VHDL y hacer buen uso de ella aquí para permitir que se cambie el tamaño del contador con un cambio de una línea (véase el ejercicio 8.53).

Tabla 8-17 Programa VHDL para un contador serie sincrónico tipo 74x163 de 8 bits.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V74x163s is
  port( CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
        D: in STD_LOGIC_VECTOR (7 downto 0);
        Q: out STD_LOGIC_VECTOR (7 downto 0);
        RCO: out STD_LOGIC );
end V74x163s;

architecture V74x163s_arch of V74x163s is
component syncsercell
  port( CLK, LDNOCLR, NOCLRORLD, CNTENP, D, CNTEN: in STD_LOGIC;
        CNTENO, Q: out STD_LOGIC );
end component;
signal LDNOCLR, NOCLRORLD: STD_LOGIC; -- señales comunes
signal SCNTEN: STD_LOGIC_VECTOR (8 downto 0); -- entradas de habilitación de conteo serie
begin
  LDNOCLR <= (not LD_L) and CLR_L; -- crear controles comunes de carga y borrado
  NOCLRORLD <= LD_L and CLR_L;
  SCNTEN(0) <= ENT; -- habilitación de conteo serie en la primera etapa
  RCO <= SCNTEN(8);  -- RCO es equivalente a salida de habilitación de conteo final
  g1: for i in 0 to 7 generate     -- generar las ocho etapas syncsercell
        U1: syncsercell port map ( CLK, LDNOCLR, NOCLRORLD, ENP, D(i), SCNTEN(i),
                                   SCNTEN(i+1), Q(i));
  end generate;
end V74x163s_arch;
```

8.5 Registros de corrimiento

8.5.1 Estructura del registro de corrimiento

registro de corrimiento

Un *registro de corrimiento* es un registro de *n* bits con una disposición para recorrer sus datos almacenados por una posición de bit en cada tic del reloj. La figura 8-46 ilustra la estructura de un registro de corrimiento de entrada serie y salida serie. La *entrada serie*, SERIN, especifica un nuevo bit que será introducido por un extremo para cada tic del reloj. Este bit aparece en la *salida serie*, SEROUT, después de *n* tics del reloj, y se pierde un tic más tarde. De este modo, un registro de corrimiento de *n* bits de entrada serie y salida serie puede emplearse para retardar una señal en *n* tics del reloj.

entrada serie

salida serie

registro de corrimiento de entrada serie, salida paralelo

conversión serie a paralelo

Un *registro de corrimiento de entrada serie y salida paralelo*, mostrado en la figura 8-47, tiene salidas para todos sus bits almacenados, haciéndolos quedar disponibles para otros circuitos. Un registro así puede utilizarse para efectuar *conversión serie a paralelo*, como se explicará más adelante en esta sección.

registro de corrimiento de entrada en paralelo, salida serie

A la inversa, es posible construir un *registro de corrimiento de entrada en paralelo y salida serie*. La figura 8-48 ilustra la estructura general de un dispositivo de esta naturaleza. Para cada tic de reloj el registro carga nuevos datos de las entradas 1D–ND o recorre su contenido actual, dependiendo del valor de la entrada de control LOAD/SHIFT (que podría ser llamada LOAD o SHIFT_L). Internamente, el dispositivo utiliza un multiplexor de 2 entradas en cada entrada D del flip-flop para seleccionar entre los dos casos. Un registro de corrimiento de entrada en paralelo y salida serie puede ser empleado para efectuar *conversión paralelo a serie*, como se explica más adelante en esta sección.

conversión paralelo a serie

registro de corrimiento de entrada en paralelo, salida paralelo

Al proporcionar salidas para todos los bits almacenados en un registro de corrimiento de entrada en paralelo, obtenemos el *registro de corrimiento de entrada en paralelo y salida paralelo* que se muestra en la figura 8-49. Un dispositivo así es muy general para utilizarlo en cualquiera de las aplicaciones anteriores de los registros de corrimiento.

Figura 8-46 Estructura en un registro de corrimiento de entrada serie, salida serie.

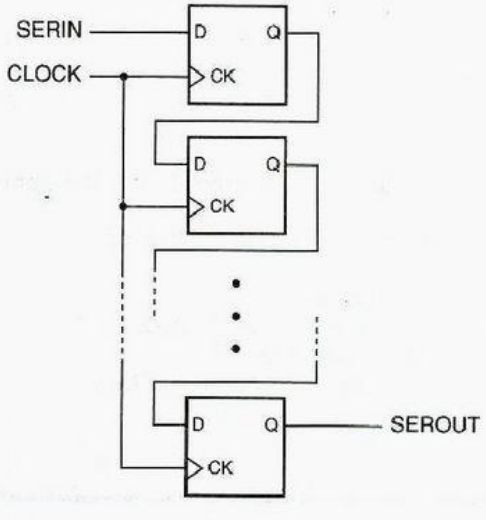
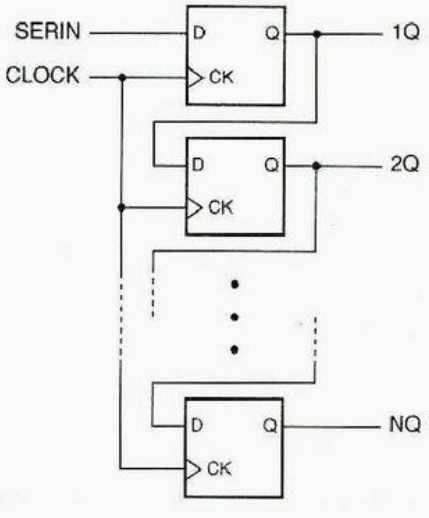


Figura 8-47 Estructura de un registro de corrimiento de entrada serie, salida paralelo.



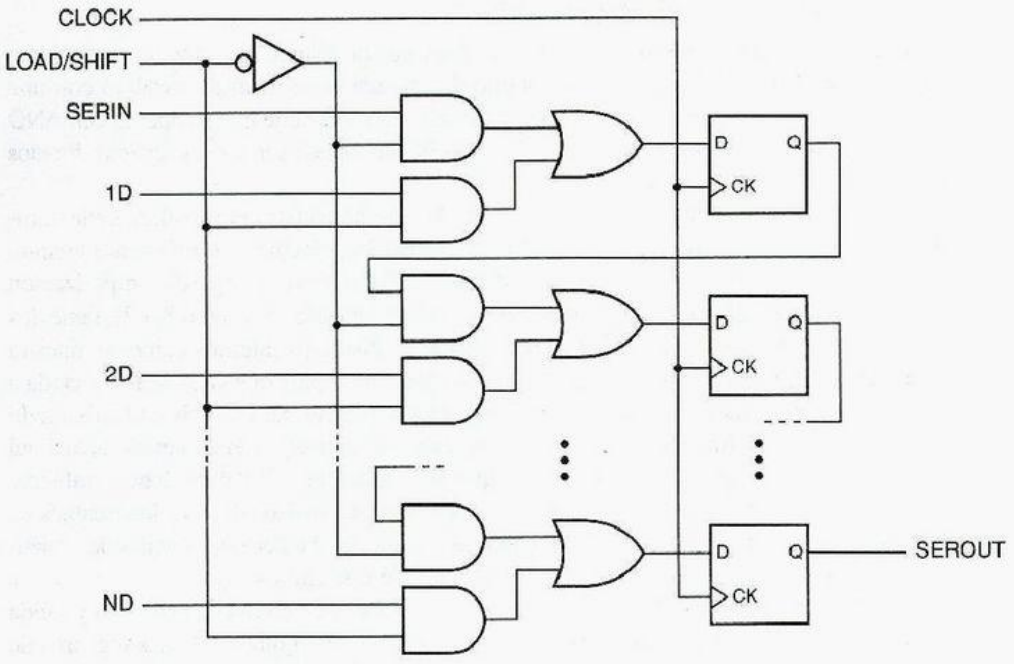
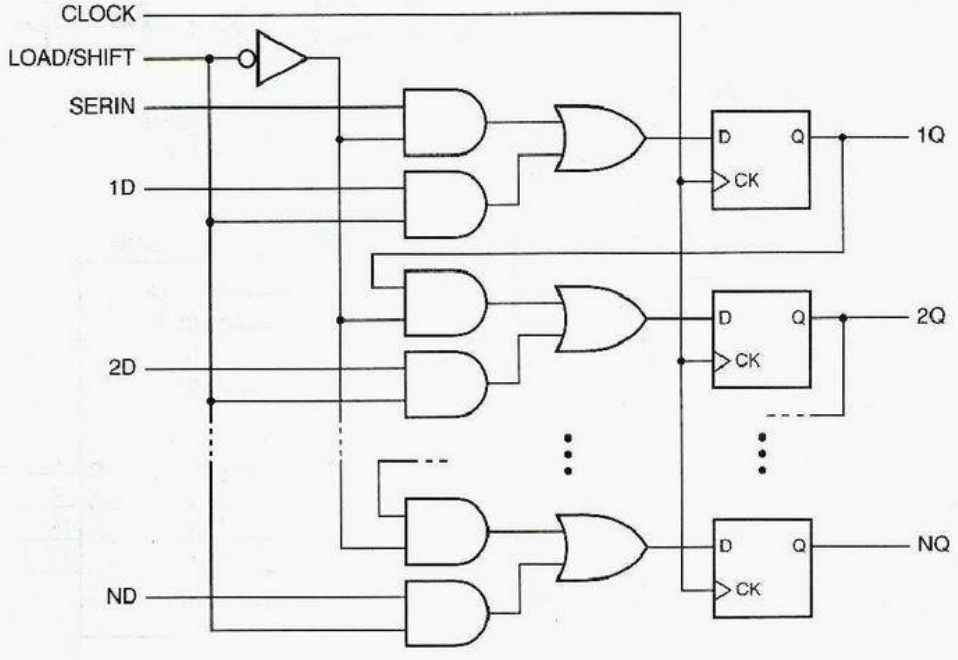


Figura 8-48 Estructura en un registro de corrimiento de entrada en paralelo, salida serie.

Figura 8-49 Estructura de un registro de corrimiento de entrada en paralelo, salida paralelo.



8.5.2 Registros de corrimiento MSI

74x164

La figura 8-50 muestra los símbolos lógicos para tres populares registros de corrimiento de 8 bits MSI. El *74x164* es un dispositivo de entrada serie y salida paralelo con una entrada de borrado asincrónica (CLR_L). Tiene dos entradas serie que se operan con AND de manera interna. Es decir, tanto SERA como SERB deben ser 1 para ser introducidos en el primer bit del registro.

74x166

El *74x166* es un registro de corrimiento de entrada en paralelo y salida serie, también con entrada asincrónica de borrado. El dispositivo efectúa el corrimiento cuando SH/LD es 1 y de otro modo carga nuevos datos. El '116 tiene un arreglo de temporización poco habitual llamado un "reloj de compuerta" (véase también la sección 8.8.2); tiene dos entradas de reloj que se encuentran conectadas a los flip-flops internos como se muestra en la figura 8-50(c). Los diseñadores del '166 lo destinaron para que CLK sea conectada a un reloj de funcionamiento libre y para CLKINH sea asertiva para inhibir CLK, de modo que no ocurra ni corrimiento ni carga en el siguiente tic de reloj, y el contenido actual del registro se mantenga. Sin embargo, para que esto funcione, CLKINH debe cambiarse solamente cuando CLK es 1; de otro modo, se presentarán flancos de reloj indeseables en los flip-flops internos. Una manera mucho más segura de obtener una función de "retención" es empleada en los siguientes dispositivos que discutimos aquí.

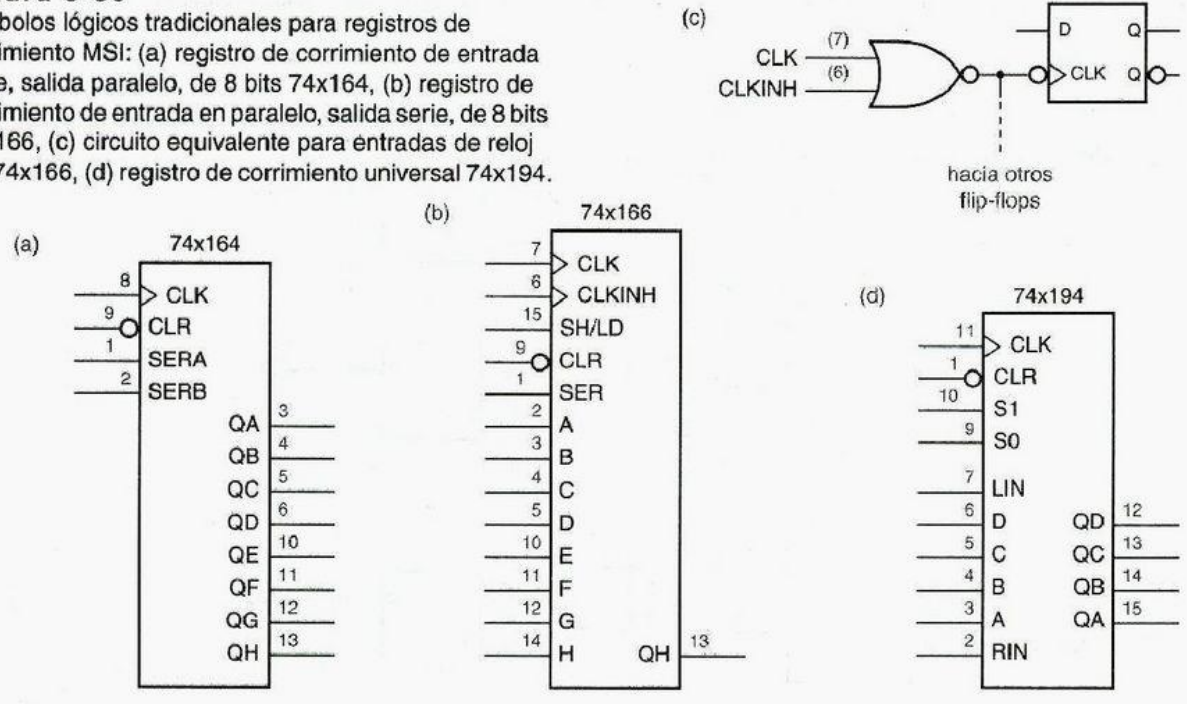
74x194

registro de corrimiento unidireccional

registro de corrimiento bidireccional

El *74x194* es un registro de corrimiento bidireccional de entrada en paralelo y salida paralelo de 4 bits MSI. Su diagrama lógico se presenta en la figura 8-51. Los registros de corrimiento que hemos estudiado con anterioridad son llamados *registros de corrimiento unidireccionales* debido a que efectúan el corrimiento en una sola dirección. El '194 es un *registro de corrimiento bidireccional* porque su contenido puede ser desplazado en cual-

Figura 8-50
Símbolos lógicos tradicionales para registros de corrimiento MSI: (a) registro de corrimiento de entrada serie, salida paralelo, de 8 bits 74x164, (b) registro de corrimiento de entrada en paralelo, salida serie, de 8 bits 74x166, (c) circuito equivalente para entradas de reloj del 74x166, (d) registro de corrimiento universal 74x194.



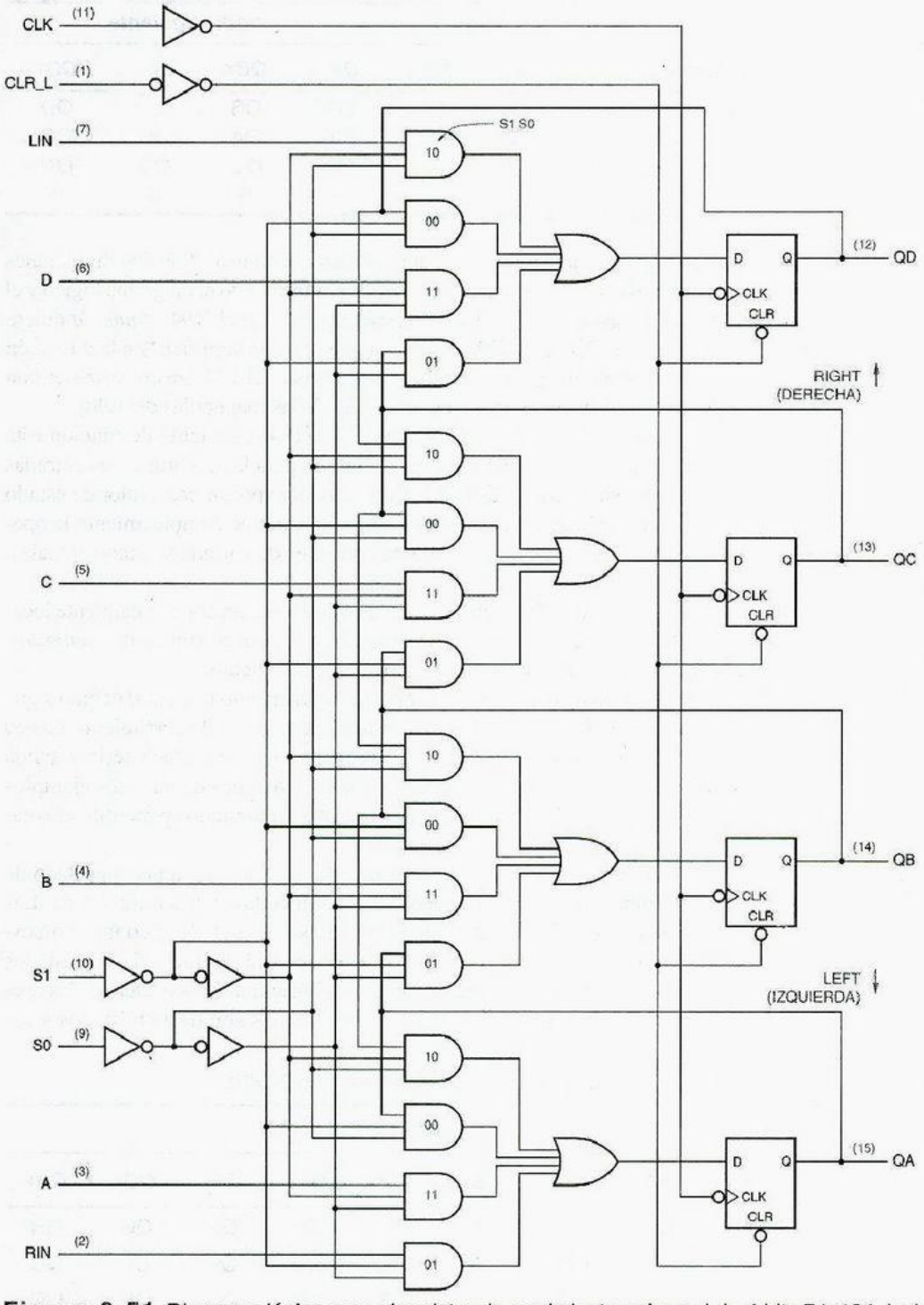


Figura 8-51 Diagrama lógico para el registro de corrimiento universal de 4 bits 74x194, incluyendo números de terminales para un encapsulado en doble línea de 16 terminales estándar.

Tabla 8-18
Tabla de función para el registro de corrimiento universal de 4 bits 74x194.

Función	Entradas S1	Entradas S0	Estado siguiente QA*	QB*	QC*	QD*
Retención	0	0	QA	QB	QC	QD
Corrimiento a la derecha	0	1	RIN	QA	QB	QC
Corrimiento a la izquierda	1	0	QB	QC	QD	LIN
Carga	1	1	A	B	C	D

izquierda
derecha

quiera de dos direcciones, dependiendo de una entrada de control. Las dos direcciones son llamadas "izquierda" ("left") y "derecha" ("right"), aun cuando el diagrama lógico y el símbolo lógico no necesariamente se dibujen de esta manera. En el '194, *izquierda* quiere decir "en la dirección de QD hacia QA", mientras que *derecha* significa "en la dirección de QA hacia QD". Nuestro diagrama y símbolo lógicos para el 194 son consistentes con estos nombres si usted los hace girar 90° en el sentido de las manecillas del reloj.

La tabla 8-18 es una tabla de función para el 74x194. Esta tabla de función está altamente comprimida, puesto que no contiene columnas para la mayoría de las entradas (A–D, RIN, LIN) o el estado actual QA-QD. Con todo, al expresar cada valor de estado siguiente como una función de estas variables implícitas, define completamente la operación del '194 para todas las 2^{12} combinaciones posibles de entrada y estado actuales, ¡y seguro que supera una tabla de 4,096 renglones!

Advierta que la entrada LIN ("left-in") del '194 se encuentra conceptualmente localizada en el lado "derecho" del chip, pero es la entrada serie para corrimientos *izquierdos*. Igualmente, RIN es la entrada serie para los corrimientos derechos.

El '194 se denomina en ocasiones un registro de corrimiento *universal* debido a que puede hacerse funcionar como cualquiera de los tipos de registro de corrimiento menos generales que hemos discutido (por ejemplo, unidireccional; de entrada serie y salida paralelo; de entrada en paralelo y salida serie). De hecho, muchos de nuestros ejemplos de diseño en las siguientes subsecciones contienen '194 configurados para utilizar solamente un subconjunto de sus funciones disponibles.

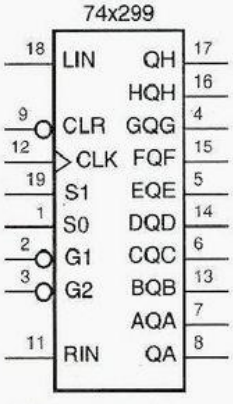


Figura 8-52
Símbolo lógico tradicional para el 74x299.

El *74x299* es un registro de corrimiento universal de 8 bits en un encapsulado de 20 terminales; su símbolo y diagrama lógicos se ilustran en las figura 8-52 y 8-53. Las funciones y la tabla de función del '299 son semejantes a las del '194, como se muestra en la tabla 8-19. Para ahorrar terminales, el '299 hace uso de líneas de tres estados bidireccionales para entrada y salida, como se ilustra en el diagrama lógico. Durante las operaciones de carga (S1 S0 = 11), los buffers de tres estados son deshabilitados y los

Tabla 8-19 Tabla de función para un registro de corrimiento universal de 8 bits 74x299.

Función	Entradas S1	Entradas S0	Estado siguiente QA*	QB*	QC*	QD*	QE*	QF*	QG*	QH*
Retención	0	0	QA	QB	QC	QD	QE	QF	QG	QH
Corrimiento a la derecha	0	1	RIN	QA	QB	QC	QD	QE	QF	QG
Corrimiento a la izquierda	1	0	QB	QC	QD	QE	QF	QG	QH	LIN
Carga	1	1	AQA	BQB	CQC	DQD	EQE	FQF	GQG	HQH

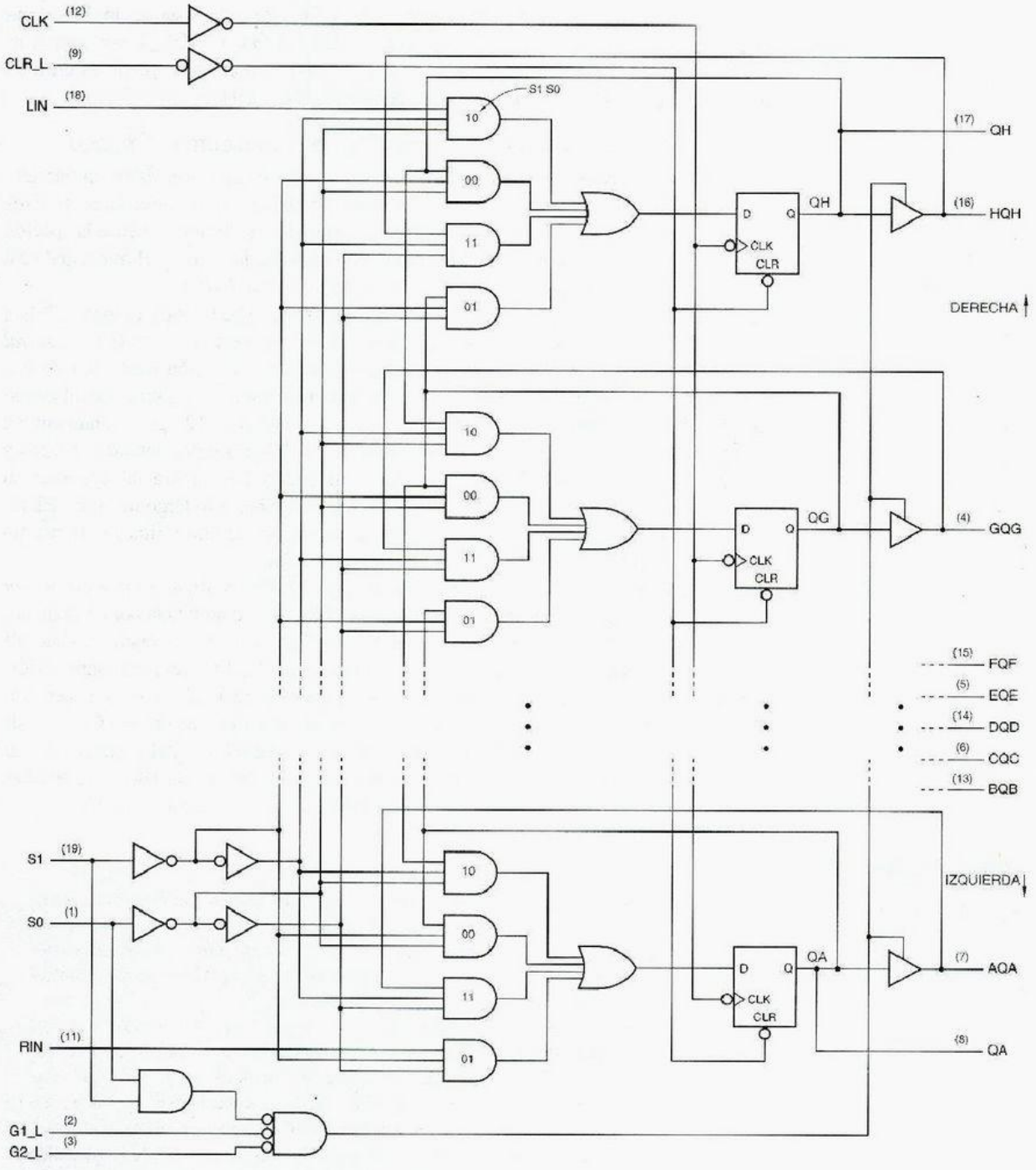


Figura 8-53 Diagrama lógico para el registro de corrimiento universal de 8 bits 74x299, incluyendo números de terminales para un encapsulado en doble línea de 20 terminales estándar.

datos son cargados a través de las terminales AQA–HQH. En otros instantes los bits almacenados son colocados sobre estas mismas terminales si G1_L y G2_L son asertivas. Los bits almacenados en los extremos izquierdo y derecho están disponibles en todo momento en terminales sólo de salida por separado, QA y QH.

8.5.3 La más grande aplicación de registro de corrimiento del mundo

La aplicación más común de los registros de corrimiento es convertir datos en paralelo al formato serie para transmisión o almacenamiento, así como la conversión de datos serie de vuelta al formato en paralelo para procesamiento o exhibición (véase la sección 2.16.1). El ejemplo más común de transmisión de datos serie, uno en el que usted casi con toda certeza ha tomado parte a diario, es en la *telefonía digital.*

telefonía digital

CO

Por años, las compañías telefónicas (TPC, del inglés "The Phone Companies") han estado instalando equipo de conmutación digital en sus oficinas centrales (*CO*, "*central offices*"). La mayoría de los teléfonos caseros tienen una conexión analógica de dos alambres hacia la oficina central. Sin embargo, un convertidor analógico a digital muestrea la señal de voz analógica 8,000 veces por segundo (una cada 125 μs) cuando entra a la CO y produce una secuencia correspondiente de 8,000 bytes representando el signo y amplitud de la señal analógica en cada punto de muestreo. De manera subsiguiente, su voz es transmitida digitalmente en *canales en serie* de 64 Kbps a lo largo de la red telefónica, hasta que se convierte de nueva cuenta en una señal analógica mediante un convertidor digital a analógico en la CO, al otro extremo.

canal serie

multiplexor

El ancho de banda de 64 Kbps requerido por una señal de voz digital es bastante *menor* que el que puede obtenerse en una sola línea de señal digital o conmutada por CI digitales. Por consiguiente la mayor parte del equipo telefónico digital *multiplexa* muchos canales de 64 Kbps en un solo alambre, ahorrando tanto alambres como CI digitales para conmutación. En la subsección siguiente, mostraremos cómo se pueden procesar 32 canales con la ayuda de pocos chips MSI; y estos chips podrían ser fácilmente integrados en un solo CPLD. Éste es un ejemplo clásico de un *compromiso espacio/temporal* en diseño digital: al funcionar más rápido los chips, se puede llevar a cabo una tarea más grande con menos chips. En realidad, ésta es la razón principal por la cual la red telefónica se haya "hecho digital".

intercambio espacio/tiempo

(I STILL DON´T KNOW)

La tecnología ISDN (Integrated Services Digital Network, red digital de servicios integrados) fue desarrollada a finales de 1980 para extender los canales digitales en serie de 144 Kpbs dúplex completo a los aparatos telefónicos caseros. La idea era transportar dos conversaciones de voz de 64 Kbps más un canal de control de 16 Kbps sobre un par simple de alambres, incrementando de este modo la capacidad del alambrado instalado.

En la primera edición de este libro, advertimos que los retardos en la utilización del ISDN habían llevado a algunas personas en la industria a renombrarlo como "Imaginary Services Delivered Nowhere" ("Servicios imaginarios entregados en ningún lugar"). A mediados de 1990, ISDN finalmente despegó en los Estados Unidos de Norteamérica, pero no fue tan utilizado para transportar voces como para proporcionar conexiones de "alta velocidad" para Internet.

Desgraciadamente para las compañías telefónicas, el desarrollo en ISDN fue detenido primero por el despliegue de los económicos modernos analógicos de 56 Kbps y posteriormente por la creciente disponibilidad de conexiones de muy alta velocidad (de 160 Kbps hasta 2 Mbps o mayores) utilizando las más novedosas tecnologías, DSL (*Digital Subscriber Line*, línea digital de suscriptor) y módem por cable.

8.5.4 Conversión serie/paralela

Una aplicación típica de la transferencia de datos serie entre dos módulos (posiblemente en un segmento de equipo de conmutación de CO) se muestra en la figura 8-54. Normalmente se conectan tres señales entre un módulo fuente y un módulo de destino para llevar a cabo la transferencia:

- *Reloj.* La señal de reloj proporciona la referencia de temporización para transferencias, definiendo el tiempo para transferir un bit. En sistemas con sólo dos módulos, el reloj puede ser parte de los circuitos de control localizados en el módulo fuente como se muestra. En sistemas más grandes, el reloj puede ser generado en un punto común y distribuido a todos los módulos.
- *Datos en serie.* Los datos mismos se transmiten en una sola línea.
- *Sincronización.* Un *pulso de sincronización* (o *pulso de sincronía, "sync"*) proporciona un punto de referencia para definir el formato de los datos, tal como el principio de un byte o palabra en el flujo de datos en serie. Algunos sistemas omiten esta señal y en su lugar utilizan un patrón único sobre la línea de datos en serie para la sincronización.

pulso de sincronización
pulso de sincronía (sync)

Las características de temporización generales de estas señales en una aplicación telefónica digital típica se muestran en la figura 8-55(a). La señal de reloj (CLOCK) tiene una frecuencia de 2.048 MHz para permitir la transferencia de $32 \times 8{,}000$ bytes de 8 bits por segundo. El pulso de 1 bit de ancho en la señal SYNC identifica el comienzo

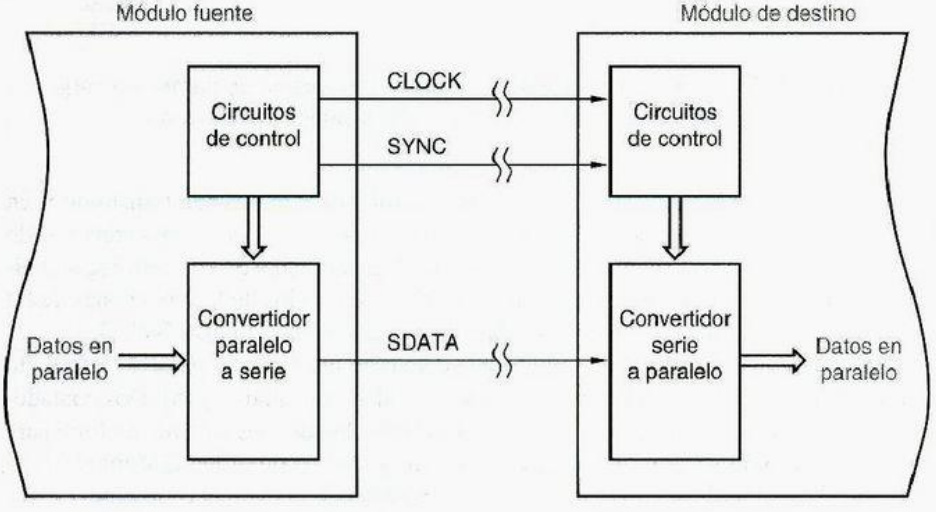


Figura 8-54
Un sistema que transmite datos de manera serial entre módulos.

EL RELOJ DE LA NACIÓN

¡Créalo o no, en la red telefónica, un reloj muy preciso de 8 kHz se genera en St. Louis y se distribuye a lo largo y ancho de los Estados Unidos! La señal de reloj que es distribuida en un segmento particular de equipo de CO local normalmente se obtiene el reloj nacional. Por ejemplo, el reloj de 2,048 MHz en el ejemplo de esta sección podría obtenerse mediante un circuito de lazo de amarre de fase (PLL) que multiplique la frecuencia del reloj nacional por 256.

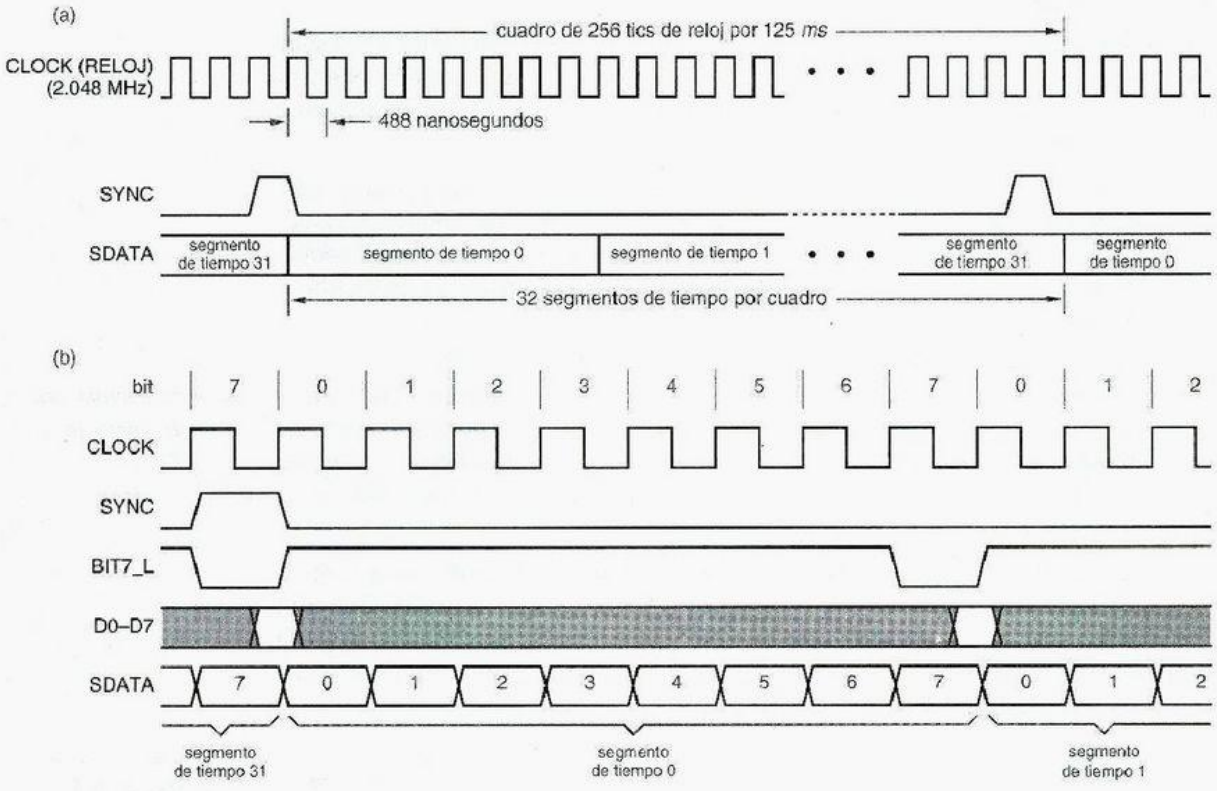


Figura 8-55 Diagrama de temporización para la conversión de paralelo a serie: (a) un cuadro completo; (b) un byte al principio del cuadro.

cuadro
segmento de tiempo

de un intervalo de 125 μs llamado un *cuadro*. Un total de 256 bits son transmitidos en SDATA durante este intervalo, el cual se divide en 32 *segmentos de tiempo* conteniendo cada uno ocho bits. Cada segmento de tiempo conduce una señal de voz codificada digitalmente. Tanto los números de segmentos de tiempo como los de las posiciones de bit dentro de un segmento de tiempo se localizan en relación con el pulso SYNC.

La figura 8-56 muestra un circuito que convierte los datos en paralelo a formato serie de la figura 8-55(a), con la temporización detallada mostrada en (b). Dos contadores 74x163 son alambrados como un contador módulo 256 de funcionamiento libre para definir el cuadro. Los cinco bits de orden superior y los tres de orden inferior del contador son el número de segmentos de tiempo y número de bit, respectivamente.

¿CUÁL BIT PRIMERO? La mayor parte de los enlaces en serie reales para voz digitalizada en realidad transmiten el bit 7 primero, debido a que éste es el primer bit generado por el convertidor analógico a digital que digitaliza la señal de voz. Sin embargo, para simplificar nuestros ejemplos, transmitimos primero el bit 0 de modo que el estado del contador sea igual al número de bit.

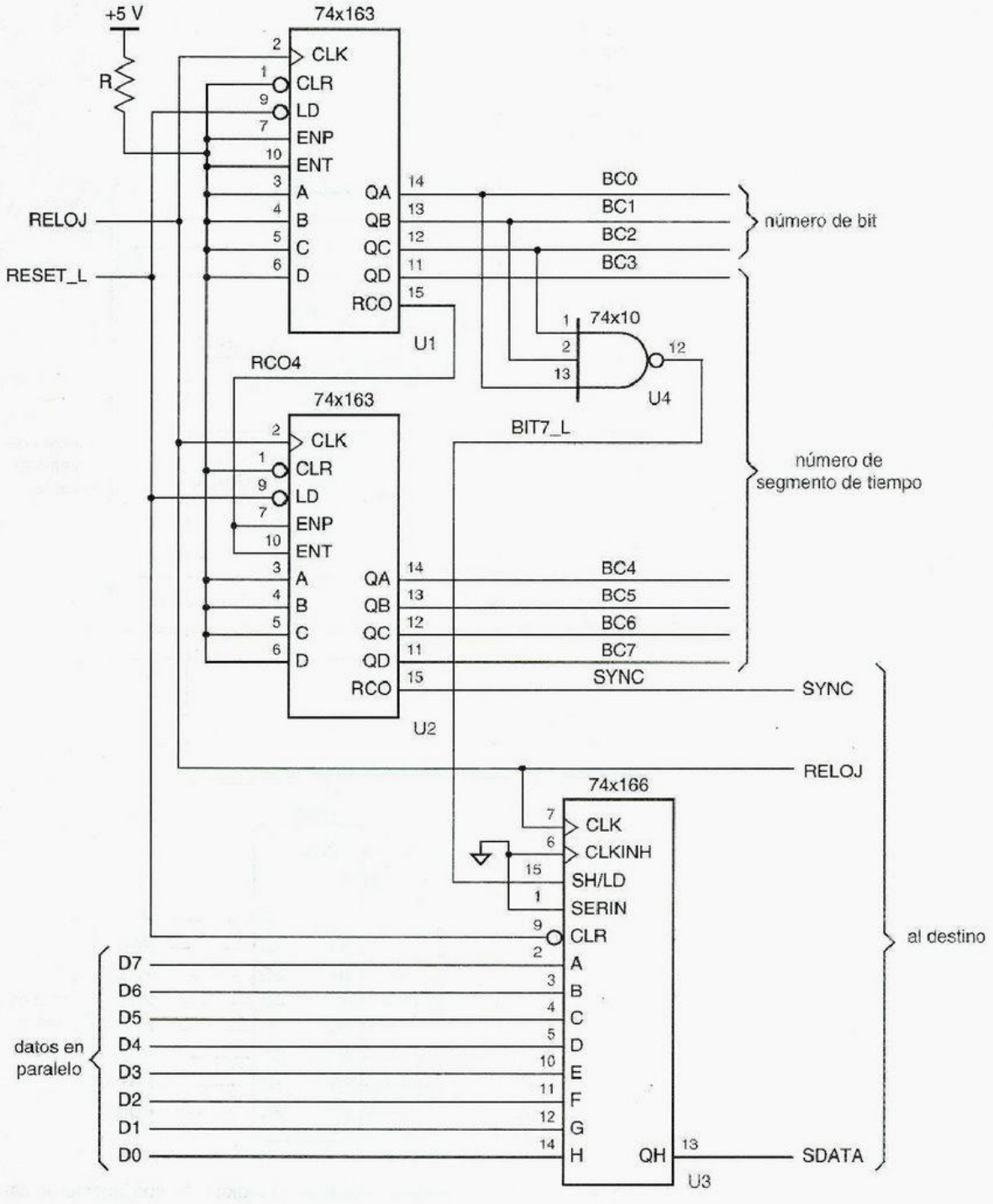


Figura 8-56 Conversión paralelo a serie utilizando un registro de corrimiento de entrada en paralelo.

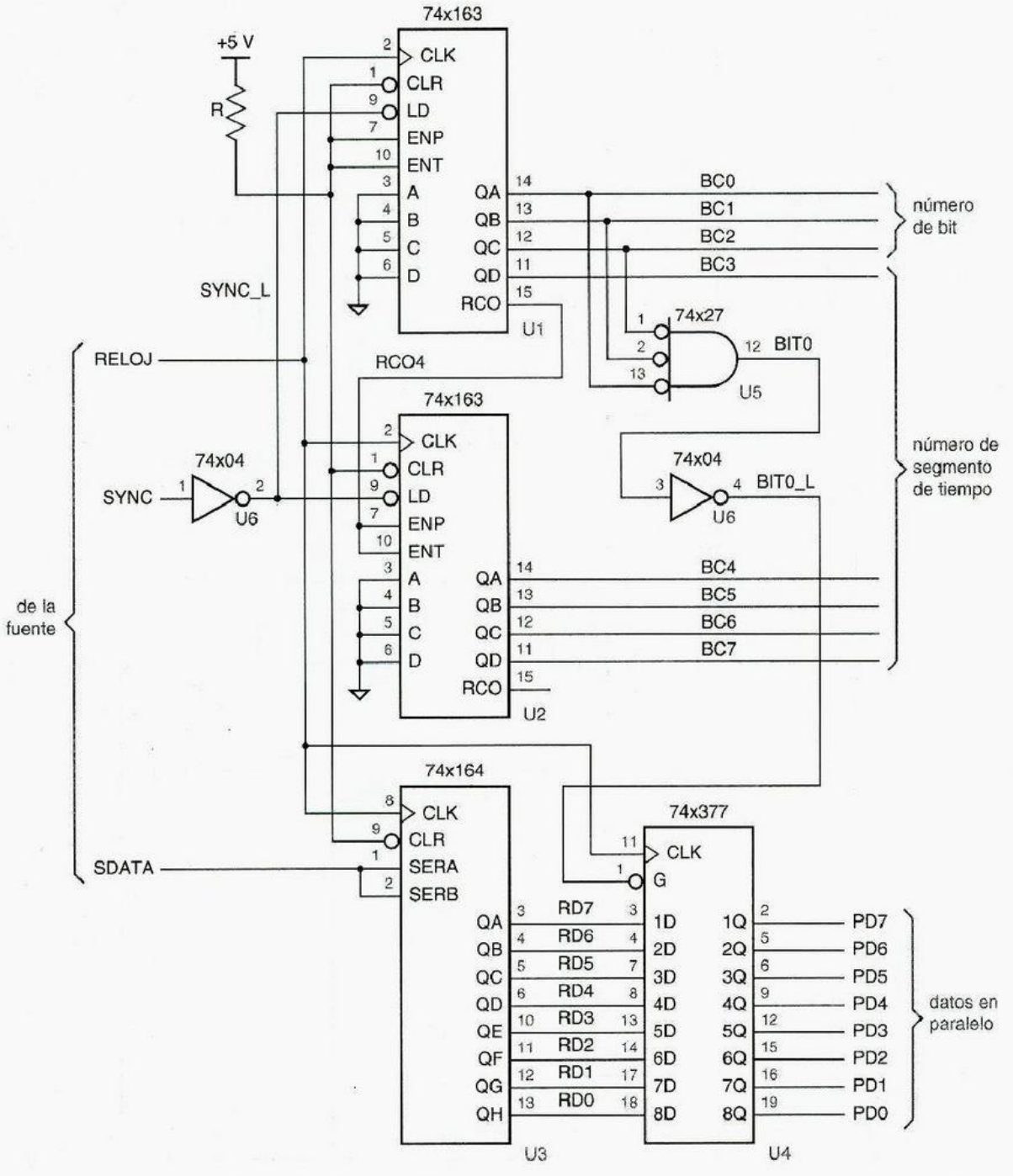


Figura 8-57 Conversión serie a paralelo utilizando un registro de corrimiento de salida en paralelo.

Un registro de corrimiento de entrada en paralelo 74x166 tiene a su cargo la conversión de paralelo a serie. El bit 0 de los datos en paralelo (D0-D7) está conectado a la entrada del '166 más cercana a la salida SDATA, de modo que los bits son transmitidos en serie en el orden del 0 al 7.

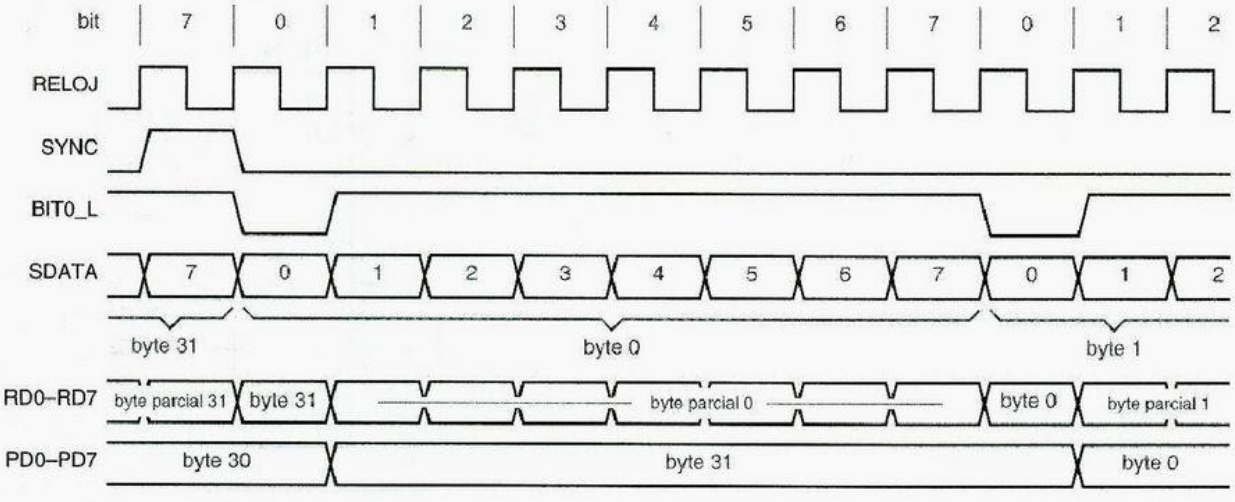
Durante el bit 7 de cada segmento de tiempo, la señal BIT7_L es asertiva, lo que provoca que el '166 sea cargado con D0–D7. El valor de D0–D7 es irrelevante excepto durante la ventana del tiempo de establecimiento y retención alrededor del flanco de reloj en el que se carga el '166, como se muestra por medio de un sombreado en el diagrama de temporización. Esto deja abierta la posibilidad de que el bus de datos en paralelo pueda ser empleado para otras cosas en otros momentos (vea el ejercicio 8-54).

Un módulo de destino puede convertir los datos en serie de vuelta al formato en paralelo utilizando el circuito de la figura 8-57. Un contador módulo 256 construido a partir de un par de '163 se utiliza para reconstruir los segmentos de tiempo y los números de bit. Aunque SYNC es asertiva durante el estado 255 del contador en el módulo fuente, SYNC carga el contador del módulo de destino con 0 de manera que ambos contadores se van a 0 en el mismo flanco de reloj. Los bits de orden superior del contador (número de segmentos de tiempo) no se utilizan en la figura, pero se pueden utilizar en otros circuitos del módulo de destino para identificar el byte de un segmento de tiempo particular, en el bus de datos en paralelo (PD0-PD7).

La figura 8-58 muestra la temporización detallada para el circuito de conversación serie a paralelo. Un byte recibido completo se encuentra disponible en la salida en paralelo del registro de corrimiento 74x164 durante el periodo de reloj que sigue a la recepción del último bit (7) del byte. Los datos en paralelo en este ejemplo están *doblemente compensados* (tienen *doble buffer*): una vez que son completamente recibidos, se transfieren a un registro 74x377, donde se encuentran disponibles en PD0-PD7 para ocho periodos de reloj completos hasta que el siguiente byte es completamente recibido. La señal BITO_L habilita al '377 para cargarse en el momento apropiado. Podrían proporcionarse registros y codificación adicionales para cargar el byte de cada segmento de tiempo en un registro diferente, haciendo cada byte disponible para 125 μs (vea la sección 8.57).

datos doblemente compensados (con doble buffer)

Figura 8-58 Diagrama de temporización para la conversión en serie paralelo.



LITTLE ENDIANS Y BIG ENDIANS

En algún punto en la evolución de los sistemas digitales, la elección de cuál bit o byte transmitir primero era una cuestión religiosa. En un famoso artículo acerca del tema, "On Holy Wars and a Plea for Peace" (*Computer*, octubre de 1981, pág. 48-54), Danny Cohen describió las diferencias en las convenciones de ordenación de bit y de byte y los estragos que podrían ser (y ahora han sido) forjados como resultado.

Nunca fue establecido un estándar firme, de modo que hoy existen algunos populares sistemas de computadoras (tales como las PC combatibles con IBM) que transmiten o numeran el byte de menor orden de una palabra de 32 bits primero, y otros (tales como las computadoras Macintosh de Apple) que transmiten o numeran primero el byte de orden mayor. Siguiendo la nomenclatura de Cohen, las personas se refieren a estas convenciones como "Little Endian", "Gig Endian", respectivamente, y platican acerca de la "endianness" como si en realidad fuera una palabra.

Una vez que los datos recibidos se encuentran en formato paralelo, pueden fácilmente ser almacenados o modificados por otros circuitos digitales; proporcionaremos ejemplos en la sección 10.1.6. En la telefonía digital, los datos en paralelo recibidos se convierten de vuelta en un voltaje analógico que es filtrado y transmitido a un audífono o bocinas por 125 μs, hasta que la siguiente muestra de voz llega.

8.5.5 Contadores de registro de corrimiento

contador de registro de corrimiento

La conversión serie/paralela es una aplicación de "datos", pero los registros de corrimiento también tienen aplicaciones "sin datos". Un registro de corrimiento puede ser mezclado con lógica combinacional para formar una máquina de estado cuyo diagrama de estado sea cíclico. Un circuito de esta naturaleza se denomina un *contador de registro de corrimiento*. A diferencia de un contador binario, un contador de registro de corrimiento no cuenta en una secuencia binaria ascendente o descendente, pero no obstante, es útil en muchas aplicaciones de "control".

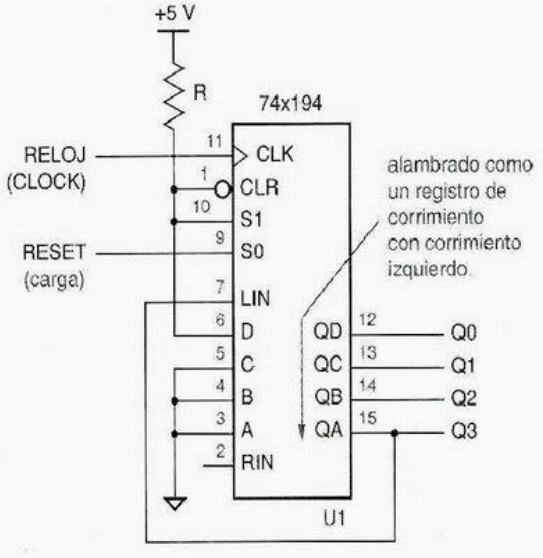


Figura 8-59
El diseño más sencillo para un contador de anillo de 4 estados, 4 bits con solo un 1 en circulación.

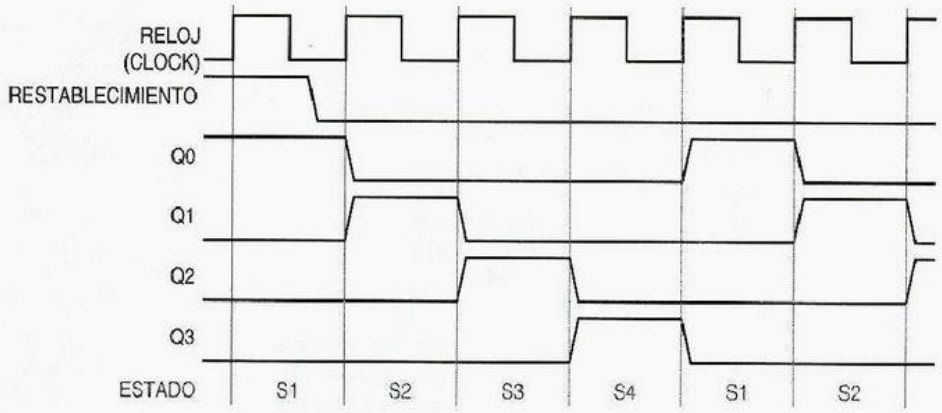


Figura 8-60
Diagrama de temporización para un contador en anillo de 4 bits.

8.5.6 Contador en anillo

contador en anillo

El contador de registro de corrimiento más sencillo utiliza un registro de corrimiento de n bits para obtener un contador con n estados, y es llamado un *contador en anillo*. La figura 8-59 es el diagrama lógico para un contador en anillo de 4 bits. El registro de corrimiento universal 74x194 es alambrado de modo que normalmente efectúe un corrimiento izquierdo. Sin embargo, cuando RESET es asertiva, carga 0001 (consulte la tabla de función del '194, tabla 8-18). Una vez que RESET es negada, el '194 se desplaza a la izquierda en cada tic del reloj. La entrada en serie LIN está conectada a la salida "más a la izquierda", de modo que los estados siguientes son 0010, 0100, 1000, 0001, 0010, ... De esta forma, el contador visita cuatro únicos estados antes de repetirse. Se muestra un diagrama de temporización en la figura 8-60. En general, un contador en anillo de n bits visita n estados en un ciclo.

El contador en anillo en la figura 8-59 tiene un problema principal: no es robusto. Si su única salida 1 se pierde debido a un problema de hardware temporal (por ejemplo, ruido), el contador se va al estado 0000 y se mantiene ahí eternamente. Igualmente, si una salida 1 extra se establece (por ejemplo, si se crea el estado 0101), el contador pasará a través de un ciclo incorrecto de estados y se quedará en ese ciclo por siempre. Estos problemas son bastante evidentes si dibujamos el diagrama de estado *completo* para el circuito contador, el cual tiene 16 estados. Como se muestra en la figura 8-61, existen 12 estados que no son parte del ciclo de conteo normal. Si el contador por alguna razón se sale del ciclo normal, se queda fuera de él.

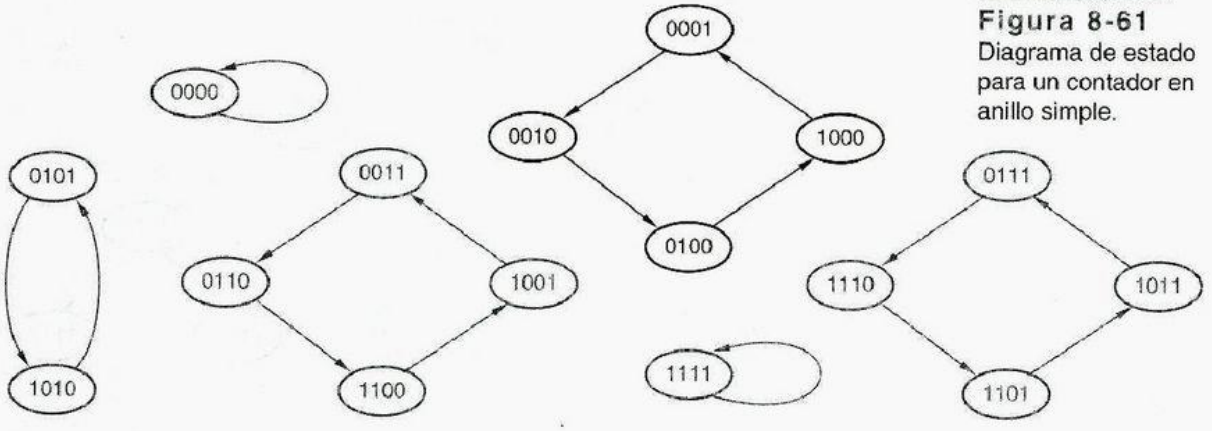


Figura 8-61
Diagrama de estado para un contador en anillo simple.

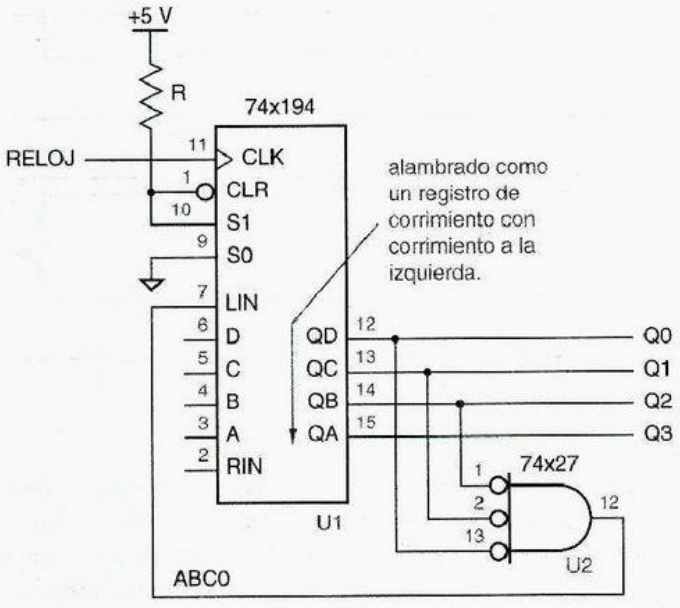


Figura 8-62
Contador en anillo de 4 estados, 4 bits, de autocorrección, con un solo 1 en circulación.

contador de autocorrección

Un *contador de autocorrección* se diseña de tal modo que todos los estados anormales tengan transiciones conduciendo a los estados normales. Los contadores de autocorrección son deseables por la misma razón por la que usamos un enfoque de mínimo riesgo para establecer asignaciones en la sección 7.4.3: si ocurriera algo inesperado, un contador o máquina de estado deberá irse a un estado "seguro".

contador en anillo con autocorrección

Un circuito *contador en anillo de autocorrección* se ilustra en la figura 8-62. El circuito utiliza una compuerta NOR para desplazar un 1 en LIN solamente cuando los tres bits menos significativos son 0. Esto resulta en el diagrama de estado de la figura 8-63;

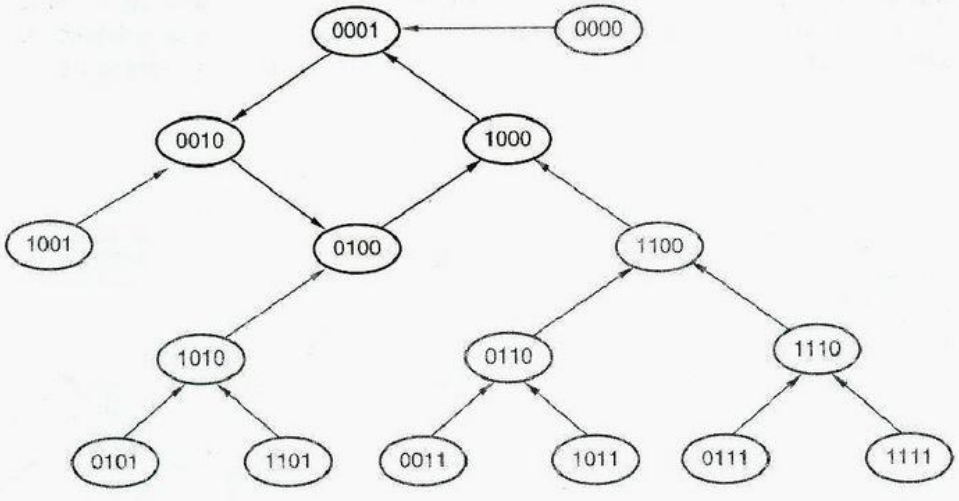


Figura 8-63
Diagrama de estado para un contador en anillo con autocorrección.

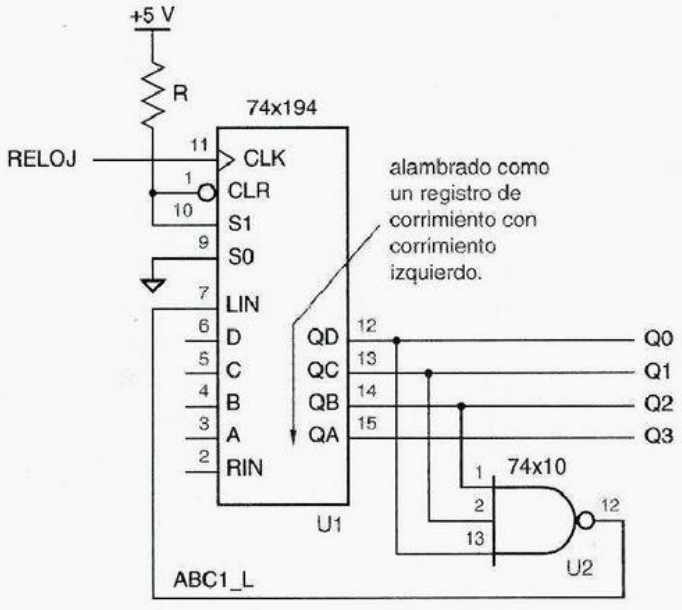


Figura 8-64
Contador en anillo de 4 estados y 4 bits con autocorrección con un 0 simple circulando.

todos los estados anormales conducen de vuelta al ciclo normal. Note que, en este circuito, una señal de RESET explícita no necesariamente es requerida. Sin tener en cuenta el estado inicial del registro de corrimiento en el momento del encendido, alcanza el estado 0001 dentro de los cuatro tics de reloj. Por lo tanto, se requiere de una señal de restablecimiento explícita solamente si es necesario asegurar que el contador arranque de manera sincrónica con otros dispositivos en el sistema o para proporcionar un punto de arranque conocido en simulación.

En el caso general, un contador en anillo de autocorrección de n bits utiliza una compuerta NOR de $(n - 1)$ entradas y corrige un estado anormal dentro de los $n - 1$ tics de reloj.

En las familias lógicas CMOS y TTL, las compuertas amplias NAND son generalmente más fáciles de conseguir que las NOR, de modo que puede ser más conveniente diseñar un contador en anillo de autocorrección como se muestra en la figura 8-64. Los estados en este ciclo normal del contador tienen un solo 0 circulando.

El principal atractivo de un contador en anillo para aplicaciones de control es que sus estados aparecen en forma decodificada de 1 de n directamente en las salidas del flip-flop. Es decir, exactamente una salida del flip-flop es asertiva en cada estado. Además, estas salidas se encuentran "libres de interferencia"; compare con el enfoque del contador y decodificador binarios de la figura 8-42.

*8.5.7 Contadores Johnson

Un registro de corrimiento de n bits con el complemento de la salida en serie alimentado de vuelta en la entrada en serie es un contador con $2n$ estados y se denomina un *contador de anillo trenzado, de Moebius o Johnson*. La figura 8-65 es el circuito básico para un contador Johnson y la figura 8-66 es su diagrama de temporización. Los estados normales

contador de anillo trenzado
contador Moebius

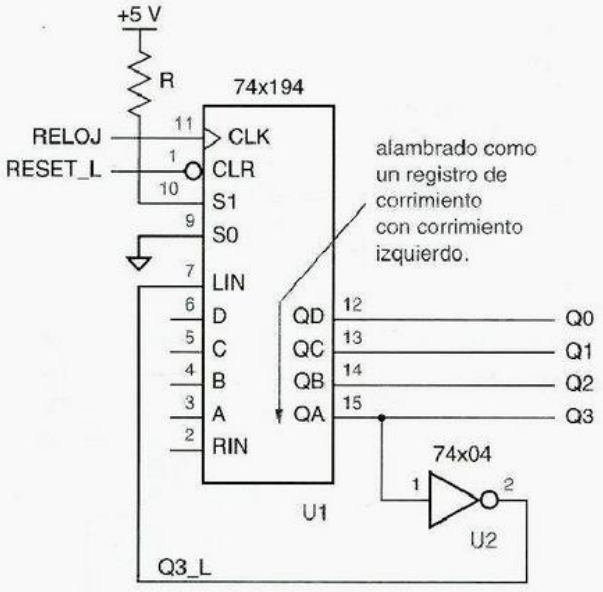


Figura 8-65
Contador Johnson básico de 8 estados y 4 bits.

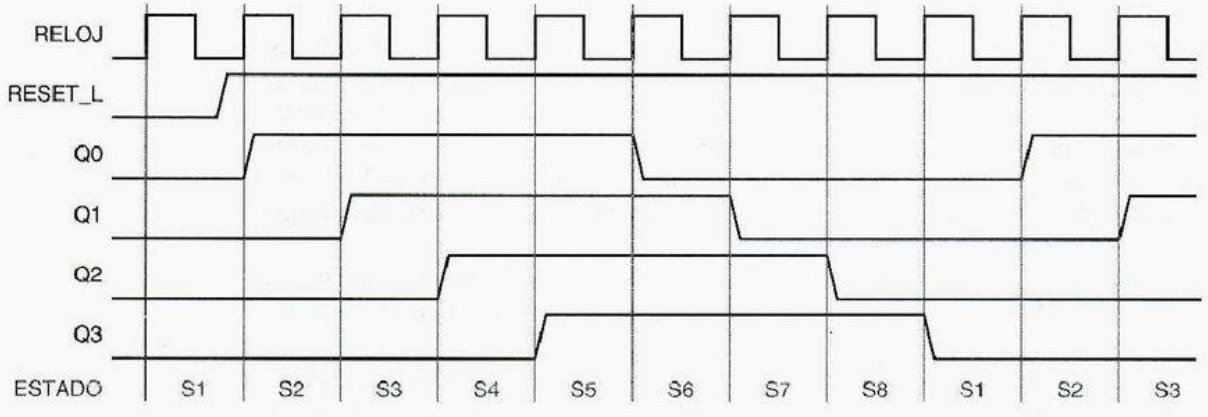


Figura 8-66 Diagrama de temporización para un contador Johnson de 4 bits.

Tabla 8-20
Estados de un contador Johnson de 4 bits.

Nombre del estado	*Q3*	*Q2*	*Q1*	*Q0*	*Decodificación*
S1	0	0	0	0	$Q3' \cdot Q0'$
S2	0	0	0	1	$Q1' \cdot Q0$
S3	0	0	1	1	$Q2' \cdot Q1$
S4	0	1	1	1	$Q3' \cdot Q2$
S5	1	1	1	1	$Q3 \cdot Q0$
S6	1	1	1	0	$Q1 \cdot Q0'$
S7	1	1	0	0	$Q2 \cdot Q1'$
S8	1	0	0	0	$Q3 \cdot Q2'$

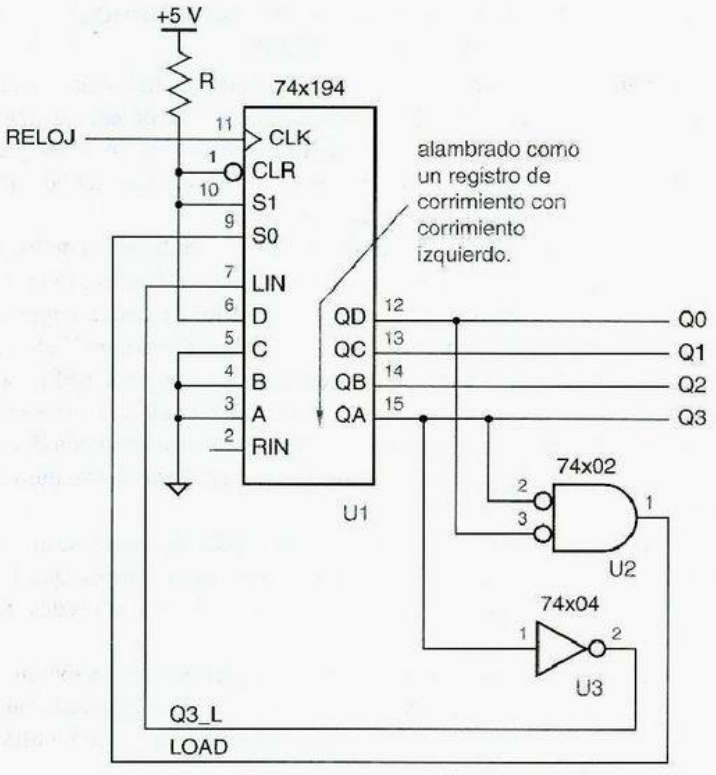


Figura 8-67
Contador Johnson de 8 estados, 4 bits, con autocorrección.

de este contador se enumeran en la tabla 8-20. Si se tienen disponibles tanto la salida verdadera como la salida complementada de cada flip-flop, cada estado normal del contador puede ser decodificado con una compuerta de 2 entradas AND o NAND, como se muestra en la tabla. Las salidas decodificadas están libres de interferencias.

contador Johnson

Un contador Johnson de n bits tiene $2^n - 2n$ estados anormales y por lo tanto está sujeto a los mismos problemas de robustez que un contador en anillo. Un *contador Johnson* de 4 bits puede diseñarse como se muestra en la figura 8-67. Este circuito carga 0001 como el siguiente estado cada vez que el estado actual sea 0xx0. Un circuito similar usando una sola compuerta NOR de 2 entradas puede realizar la corrección para un contador Jonson con cualquier número de bits. El circuito de corrección debe cargar 00...01 como el siguiente estado cada vez que el estado actual sea 0x...x0.

contador Johnson con autocorrección

¡EL CIRCUITO CON AUTOCORRECCIÓN ES CORRECTO ÉL MISMO!

Podemos probar que el circuito con autocorrección del contador Johnson corrige cualquier estado anormal como sigue. Un estado anormal siempre puede escribirse en la forma x...x10x...x, puesto que los únicos estados que no pueden escribirse en esta forma son los estados normales (00...00, 11...11, 01...1, 0...01...1, y 0...01). Por lo tanto, dentro de los $n - 2$ tics de reloj, el registro de corrimiento contendrá 10x...x. Un tic más tarde, contendrá 0x...x0, y otro tic después el estado normal 00...01 será cargado.

*8.5.8 Contadores de registro de corrimiento con retroalimentación lineal

Los contadores de registro de corrimiento de n bits que hemos mostrado hasta ahora tienen mucho menos del máximo de 2^n estados normales. Un *contador de registro de corrimiento de retroalimentación lineal (LFSR, linear feedback shift-register)* de n bits puede tener $2^n - 1$ estados, casi el máximo. Un contador de esta clase se denomina con frecuencia un *generador de secuencia de longitud máxima.*

generador de secuencia de longitud máxima

campos finitos

El diseño de contadores LFSR está basado en la teoría de los *campos finitos*, que fue desarrollada por el matemático francés Évariste Galois (1811-1832) poco antes de que fuera asesinado en un duelo por un adversario político. La operación de un contador LFSR corresponde a las operaciones en un campo finito con 2^n elementos.

La figura 8-68 muestra la estructura de un contador LFSR de n bits. La entrada serie del registro de corrimiento se encuentra conectada a la suma módulo 2 de un cierto conjunto de bits de salida. Estas conexiones de retroalimentación determinan la secuencia de estado del contador. Por convención, las salidas siempre son numeradas y desplazadas en la dirección mostrada.

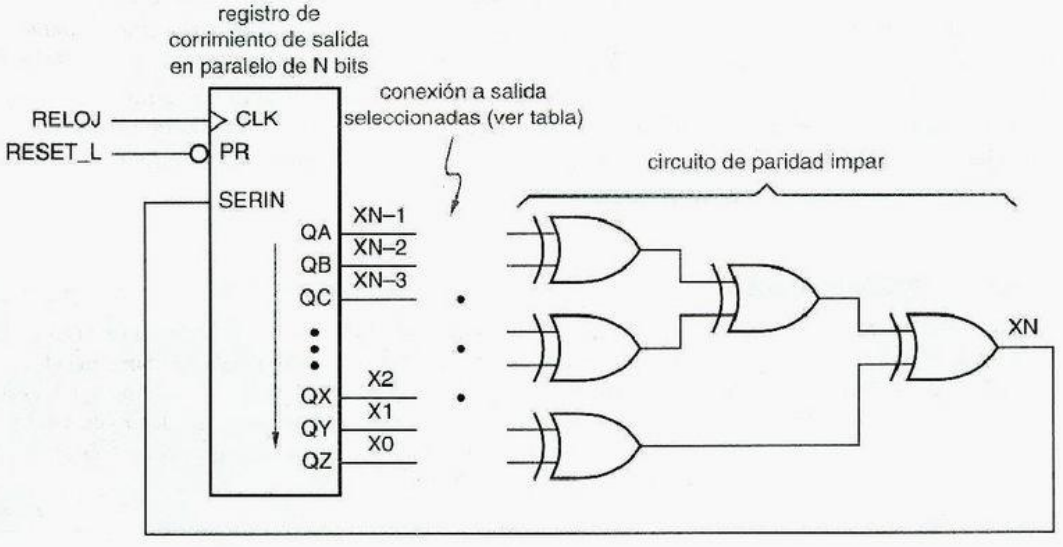
Aplicando la teoría de campos finitos, puede demostrarse que para cualquier valor de n, existe por lo menos una ecuación de retroalimentación que hace que el contador vaya a través de todos los $2^n - 1$ estados distintos de cero antes de repetir. Esto se conoce como *secuencia de longitud máxima.*

secuencia de longitud máxima

La tabla 8-21 enumera las ecuaciones de retroalimentación que resultan en secuencias de longitud máxima para valores seleccionados de n. Para cada valor de n mayor que 3, existen muchas otras ecuaciones de retroalimentación que resultan en secuencias de longitud máxima, todas diferentes.

Un contador LFSR diseñado de acuerdo a la figura 8-68 nunca puede pasar cíclicamente a través de todos los 2^n posibles estados. Sin tener en cuenta el patrón de conexión, el estado siguiente para el estado con la totalidad de ceros es el mismo: todos ceros.

Figura 8-68 Estructura general de un contador con registro de corrimiento de retroalimentación lineal.



n	*Ecuación de retroalimentación*
2	X2 = X1 ⊕ X0
3	X3 = X1 ⊕ X0
4	X4 = X1 ⊕ X0
5	X5 = X2 ⊕ X0
6	X6 = X1 ⊕ X0
7	X7 = X3 ⊕ X0
8	X8 = X4 ⊕ X3 ⊕ X2 ⊕ X0
12	X12 = X6 ⊕ X4 ⊕ X1 ⊕ X0
16	X16 = X5 ⊕ X4 ⊕ X3 ⊕ X0
20	X20 = X3 ⊕ X0
24	X24 = X7 ⊕ X2 ⊕ X1 ⊕ X0
28	X28 = X3 ⊕ X0
32	X32 = X22 ⊕ X2 ⊕ X1 ⊕ X0

Tabla 8-21
Ecuaciones de retroalimentación para los contadores con registro de corrimiento de retroalimentación lineal.

TRABAJANDO EN EL CAMPO

Un campo finito tiene un número finito de elementos y dos operadores, de suma y de multiplicación, que satisfacen ciertas propiedades. Un ejemplo de un campo finito con P elementos, donde P es cualquier número primo, es el conjunto de enteros módulo P. Los operadores en este campo son la suma y la multiplicación módulo P.

De acuerdo con la teoría de campos finitos, si usted comienza con un elemento distinto de cero E y multiplica repetidas veces por un elemento "primitivo" α, después de $P-2$ pasos usted generará el resto de los elementos distintos de cero del campo en el campo antes de llegar de vuelta a E. Resulta que en un campo con P elementos, cualquier entero en el intervalo $2, \ldots, P-1$ es primitivo. Usted puede intentar esto por sí mismo utilizando $P = 7$ y $\alpha = 2$, por ejemplo. Los elementos del campo son 0, 1, ..., 6, y las operaciones son la suma y la resta módulo 7.

El párrafo anterior de la idea básica detrás de los generadores de secuencia de longitud máxima. Sin embargo, para aplicarlos a un circuito digital, se necesita un campo con 2^n elementos, donde n es el número de bits requeridos por la aplicación. Por un lado, somos afortunados, porque Galois probó que existen campos finitos con P^n elementos para cualquier entero n, mientras P es primo, incluyendo $P = 2$. Por el otro lado, no somos afortunados, porque cuando $n > 1$, los operadores en campos con P^n (incluyendo 2^n) elementos son bastante diferentes de la suma y multiplicación de enteros ordinarias. También, los elementos primitivos son difíciles de encontrar.

Si a usted le gustan las matemáticas, como pasa conmigo, probablemente estará fascinado con la teoría que conduce a los circuitos LFSR para generadores de secuencia de longitud máxima y otras aplicaciones; consulte las Referencias. De otro modo, usted puede seguir con confianza el enfoque de "libro de recetas" en esta sección.

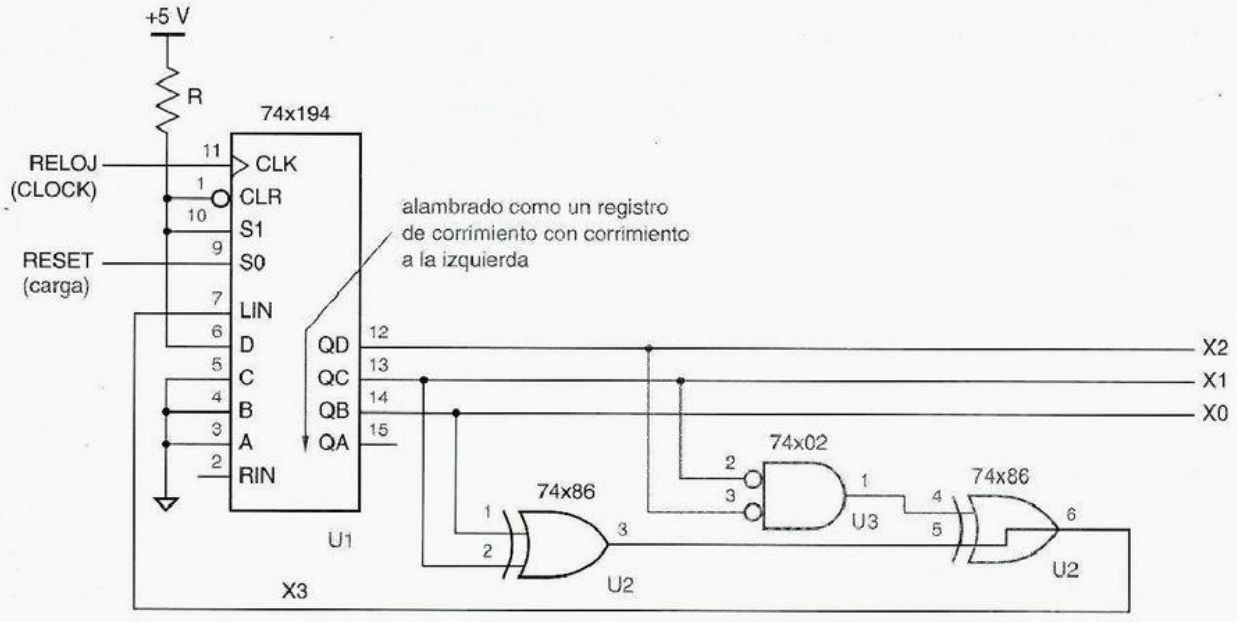


Figura 8-69 Un contador LFSR; las modificaciones para incluir el estado de ceros en su totalidad se muestran en color gris.

El diagrama lógico para un contador LFSR de 3 bits se ilustra en la figura 8-69. La secuencia de estado para este contador se muestra en las primeras tres columnas de la tabla 8-22. Comenzando en cualquier estado distinto de cero, 100 en la tabla, el contador visita siete estados antes de regresar al estado inicial.

Un contador LFSR puede ser modificado para tener 2^n estados, incluyendo el estado lleno de ceros, como se muestra en color gris para el contador de 3 bits de la figura 8-69. La secuencia de estado resultante es dada en las últimas tres columnas de la tabla 8-22. En un contador LFSR de n bits, una compuerta XOR extra y una compuerta NOR

Tabla 8-22
Secuencias de estado para el contador LFSR de 3 bits en la figura 8-69.

Secuencia original			*Secuencia modificada*		
X2	***X1***	***X0***	***X2***	***X1***	***X0***
1	0	0	1	0	0
0	1	0	0	1	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1
0	1	1	0	1	1
0	0	1	0	0	1
1	0	0	0	0	0
.	.	.	1	0	0
.	.	.	.	.	.

de $n - 1$ entradas conectadas a todas las salidas del registro de corrimiento excepto X0 lleva a cabo la misma cosa.

Los estados de un LFSR no son visitados en orden de conteo binario. Sin embargo, los contadores LFSR se utilizan en aplicaciones donde estas características son una ventaja. Una aplicación principal de los contadores LFSR es la generación de entradas de prueba para circuitos lógicos. En la mayoría de los casos, la secuencia de conteo "pseudoaleatoria" de un contador LFSR detectará probablemente más errores que una secuencia de conteo binario. Los LFSR también son empleados en los circuitos de codificación y decodificación para ciertos códigos de detección de errores y corrección de errores, incluyendo códigos CRC, los cuales presentamos en la sección 2.15.4.

En comunicaciones de datos, los contadores LFSR se utilizan con frecuencia para "cifrar" y "descifrar" los patrones de datos transmitidos por modems de alta velocidad e interfaces de red, incluyendo Ethernet de 100 Mbps. Esto se hace mediante la operación de XOR aplicada a la salida del LFSR con el flujo de datos del usuario. Aun cuando el flujo de datos del usuario contiene una secuencia larga de ceros o unos, al combinarlo con la salida pseudoleatoria del LFSR se mejora el equilibrio de DC de la señal transmitida y se crea un rico conjunto de transiciones que permite que la información registrada sea recuperada más fácilmente en el receptor.

8.5.9 Registros de corrimiento en ABEL y PLD

Los registros de corrimiento de propósito general pueden especificarse con bastante facilidad en ABEL y ajustarse bien en los PLD típicos secuenciales. Por ejemplo, la figura 8-70 y la tabla 8-23 muestran cómo realizar una función similar a la del registro de corrimiento universal 74x194 empleando un 16V8. Note que una de las terminales de E/S del 16V8, la terminal 12, se utiliza como entrada.

La realización del 16V8 para el '194 difiere del caso real únicamente en una manera: en la función de la entrada CLR_L. En el '194 real, CLR_L es una entrada asincrónica mientras que en el 16V8 es muestreada junto con otras entradas en el flanco ascendente de CLK.

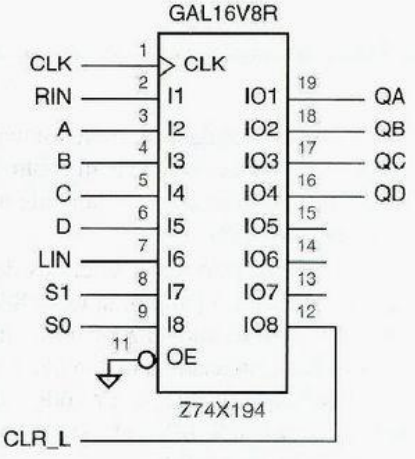


Figura 8-70
Realizaciones PLD de un registro de corrimiento universal tipo 74x194 con borrado sincrónico.

Tabla 8-23 Programa ABEL para un registro de corrimiento universal de 4 bits.

```
module Z74x194
title '4-bit Universal Shift Register'
Z74X194 device 'P16V8R';

" Input and output pins
CLK, RIN, A, B, C, D, LIN              pin 1, 2, 3, 4, 5, 6, 7;
S1, S0, CLR_L                          pin 8, 9, 12;
QA, QB, QC, QD                         pin 19, 18, 17, 16 istype 'reg';

" Active-level translation
CLR = !CLR_L;

" Set definitions
INPUT   = [  A,  B,  C, D  ];
LEFTIN  = [ QB, QC, QD, LIN];
RIGHTIN = [RIN, QA, QB, QC ];
OUT     = [ QA, QB, QC, QD ];

CTRL  = [S1,S0];
HOLD  = (CTRL == [0,0]);
RIGHT = (CTRL == [0,1]);
LEFT  = (CTRL == [1,0]);
LOAD  = (CTRL == [1,1]);

equations
OUT.CLK = CLK;

OUT := !CLR & (
           HOLD & OUT
         # RIGHT & RIGHTIN
         # LEFT & LEFTIN
         # LOAD & INPUT);

end Z74x194
```

Si usted realmente necesita proporcionar una entrada de borrado asincrónica, puede utilizar el 22V10, que proporciona una línea de producto simple para controlar las entradas de restablecimiento de todos sus flip-flops. Esto requiere solamente unos cuantos cambios en el programa original (vea el ejercicio 8.69).

La flexibilidad de ABEL puede ser empleada para crear circuitos de registro de corrimiento con mayor o diferente funcionalidad. Por ejemplo, la tabla 8-24 define un registro de corrimiento de 8 bits que puede ser borrado, cargado con un 1 simple en cualquier posición de bit, desplazado a la izquierda, desplazado a la derecha o retenido. La operación que se realiza en cada tic de reloj está especificada por un código de operación de 4 bits, OP[3:0]. A pesar del gran número de casos "WHEN", el circuito puede ser sintetizado con solamente cinco términos de producto por salida.

Tabla 8-24 Programa ABEL para un registro de corrimiento multifuncional.

```
module shifty
title '8-bit shift register with decoded load'

" Inputs and Outputs
CLK, OP3..OP0                pin;
Q7..Q0                       pin istype 'reg';

" Definitions

Q = [Q7..Q0];
OP = [OP3..OP0];

HOLD = (OP == 0);
CLEAR = (OP == 1);
LEFT = (OP == 2);
RIGHT = (OP == 3);
NOP = (OP >= 4) & (OP < 8);
LOADQ0 = (OP == 8);
LOADQ1 = (OP == 9);
LOADQ2 = (OP == 10);
LOADQ3 = (OP == 11);
LOADQ4 = (OP == 12);
LOADQ5 = (OP == 13);
LOADQ6 = (OP == 14);
LOADQ7 = (OP == 15);

Equations

Q.CLK = CLK;

WHEN HOLD THEN Q := Q;
ELSE WHEN CLEAR THEN Q := 0;
ELSE WHEN LEFT THEN Q := [Q6..Q0, Q7];
ELSE WHEN RIGHT THEN Q := [Q0, Q7..Q1];
ELSE WHEN LOADQ0 THEN Q := 1;
ELSE WHEN LOADQ1 THEN Q := 2;
ELSE WHEN LOADQ2 THEN Q := 4;
ELSE WHEN LOADQ3 THEN Q := 8;
ELSE WHEN LOADQ4 THEN Q := 16;
ELSE WHEN LOADQ5 THEN Q := 32;
ELSE WHEN LOADQ6 THEN Q := 64;
ELSE WHEN LOADQ7 THEN Q := 128;
ELSE Q := Q;

end shifty
```

Tabla 8-25 Programa para un contador en anillo de 8 bits.

```
module Ring8
title '8-bit Ring Counter'

" Inputs and Outputs
MCLK, CNTEN, RESTART                pin;
S0..S7                              pin istype 'reg';

equations

[S0..S7].CLK = MCLK;

S0 := CNTEN & !S0 & !S1 & !S2 & !S3 & !S4 & !S5 & !S6   " Self-sync
    # !CNTEN & S0                                       " Hold
    # RESTART;                                          " Start with one 1
[S1..S7] := !RESTART & ( !CNTEN & [S1..S7]      " Hold
                       # CNTEN & [S0..S6] );    " Shift
end Ring8
```

El lenguaje ABEL se puede utilizar para especificar contadores de registro de corrimiento de los diversos tipos que presentamos en las subsecciones anteriores. Por ejemplo, la tabla 8-25 es el programa para un contador en anillo de 8 bits. Hemos utilizado la capacidad extra del PLD para agregar dos funciones no presentes en nuestros anteriores diseños MSI: el conteo ocurre solamente si CNTEN es asertiva y el estado siguiente es forzado a S0 si RESTART es asertiva.

Los contadores en anillo se utilizan frecuentemente para generar relojes de fase múltiple o habilitar señales en sistemas digitales, y los requerimientos en sistemas diferentes son muchos y variados. La habilidad de reprogramar fácilmente el comportamiento del contador es una ventaja distintiva de un diseño basado en HDL.

La figura 8-71 muestra un conjunto de señales de habilitación o reloj que pueden requerirse en un sistema digital con seis distintas fases de operación. Cada fase dura dos tics de una señal maestra de reloj, MCLK, durante la cual la señal correspondiente de habilitación de fase de nivel activo bajo Pi_L es asertiva. Podemos obtener esta clase de temporización de un contador en anillo si proporcionamos un flip-flop extra para contar los dos tics de cada fase, de modo que se presente un corrimiento en el *segundo* tic de cada fase.

El generador de temporización puede ser construido con unas cuantas entradas y salidas de un PLD. Se proporcionan tres entradas de control, con el siguiente comportamiento:

RESET Cuando esta entrada es asertiva, ninguna salida es asertiva. El contador siempre va al primer tic de a fase 1 después de que RESET es negada.

RUN Cuando se afirma, esta entrada permite avanzar al contador al segundo tic de la fase actual, o al primer tic de la siguiente fase; de otro modo, el tic actual de la fase actual es extendido.

RESTART La aserción de esta entrada provoca que el contador regrese al primer tic de la fase 1, incluso si RUN no es asertiva.

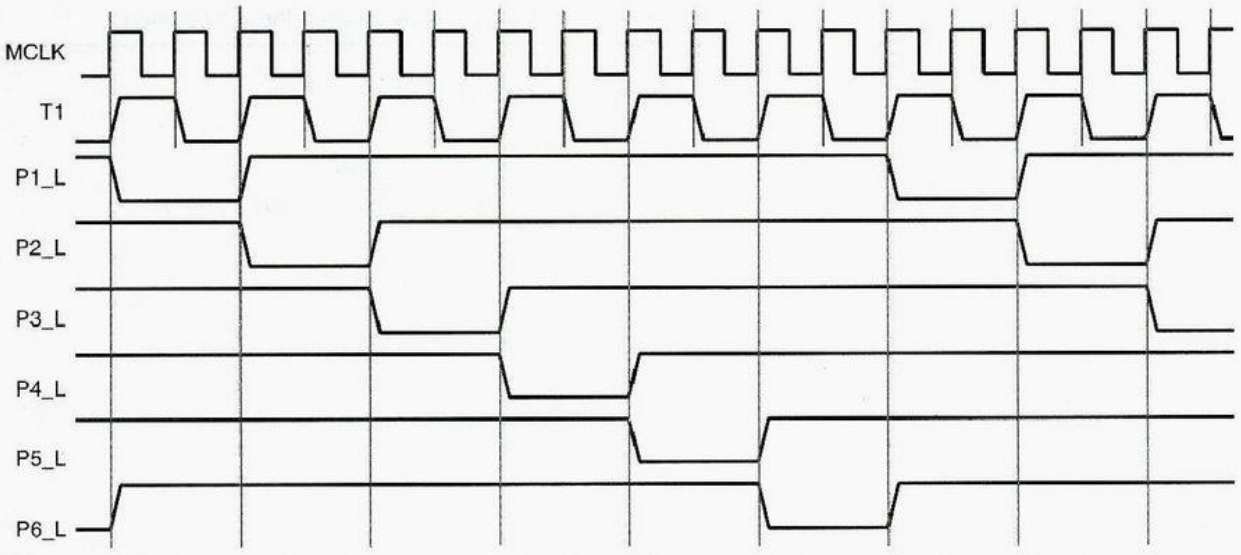


Figura 8-71 Formas de onda de temporización de seis fases requeridas en cierto sistema digital.

Tabla 8-26 Programa para un generador de formas de onda de seis fases.

```
module␣TIMEGEN6␣
title␣'Six-phase␣Master␣Timing␣Generator'

"␣Input␣and␣Output␣pins␣
MCLK,␣RESET,␣RUN,␣RESTART␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pin;
T1,␣P1_L,␣P2_L,␣P3_L,␣P4_L,␣P5_L,␣P6_L␣␣␣␣␣␣pin␣istype␣'reg';␣

"␣State␣definitions
PHASES␣=␣[P1_L,␣P2_L,␣P3_L,␣P4_L,␣P5_L,␣P6_L];
NEXTPH␣=␣[P6_L,␣P1_L,␣P2_L,␣P3_L,␣P4_L,␣P5_L];
SRESET␣=␣[1,␣1,␣1,␣1,␣1,␣1];
P1␣=␣␣␣␣␣[0,␣1,␣1,␣1,␣1,␣1];

equations
T1.CLK␣=␣MCLK;␣PHASES.CLK␣=␣MCLK;

WHEN␣RESET␣THEN␣{T1␣:=␣1;␣PHASES␣:=␣SRESET;}
ELSE␣WHEN␣(PHASES==SRESET)␣#␣RESTART␣THEN␣{T1␣:=␣1;␣PHASES␣=␣P1;}
ELSE␣WHEN␣RUN␣&␣T1␣THEN␣{T1␣:=␣0;␣PHASES␣:=␣PHASES;}
ELSE␣WHEN␣RUN␣&␣!T1␣THEN␣{T1␣:=␣1;␣PHASES␣:=␣NEXTPH;}
ELSE␣{T1␣:=␣T1;␣PHASES␣:=␣PHASES;}

end␣TIMEGEN6
```

La tabla 8-26 es un programa que crea el comportamiento requerido. Note el uso de conjuntos para especificar el comportamiento del contador en anillo de manera muy concisa, con el RESET, RESTART y RUN teniendo el comportamiento especificado en cualquier fase del contador.

Tabla 8-27 Programa alterno para el generador de formas de onda.

```
module TIMEGN6A
title 'Six-phase Master Timing Generator'

" Input and Output pins
MCLK, RESET, RUN, RESTART                    pin;
T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L       pin istype 'reg';

" State definitions
TSTATE = [T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];
SRESET = [1, 1, 1, 1, 1, 1, 1];
P1F =    [1, 0, 1, 1, 1, 1, 1];
P1S =    [0, 0, 1, 1, 1, 1, 1];
P2F =    [1, 1, 0, 1, 1, 1, 1];
P2S =    [0, 1, 0, 1, 1, 1, 1];
P3F =    [1, 1, 1, 0, 1, 1, 1];
P3S =    [0, 1, 1, 0, 1, 1, 1];
P4F =    [1, 1, 1, 1, 0, 1, 1];
P4S =    [0, 1, 1, 1, 0, 1, 1];
P5F =    [1, 1, 1, 1, 1, 0, 1];
P5S =    [0, 1, 1, 1, 1, 0, 1];
P6F =    [1, 1, 1, 1, 1, 1, 0];
P6S =    [0, 1, 1, 1, 1, 1, 0];

equations
TSTATE.CLK = MCLK;
WHEN RESET THEN TSTATE := SRESET;

state_diagram TSTATE

state SRESET: IF RESET THEN SRESET ELSE P1F;

state P1F: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P1S ELSE P1F;

state P1S: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P2F ELSE P1S;

state P2F: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P2S ELSE P2F;

state P2S: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P3F ELSE P2S;

state P3F: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P3S ELSE P3F;

state P3S: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P4F ELSE P3S;

state P4F: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P4S ELSE P4F;

state P4S: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P5F ELSE P4S;
```

Tabla 8-27 (continuación) Programa alterno para el generador de formas de onda.

```
state P5F: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P5S ELSE P5F;

state P5S: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P6F ELSE P5S;

state P6F: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P6S ELSE P6F;

state P6S: IF RESET THEN SRESET ELSE IF RESTART THEN P1F
           ELSE IF RUN THEN P1F ELSE P6S;

end TIMEGN6A
```

El mismo comportamiento del generador de temporización en la figura 8-71 puede especificarse si se utiliza un enfoque de diseño de máquina de estado, como se muestra en la tabla 8-27. Este programa ABEL, aunque es extenso, genera el mismo comportamiento externo durante operación normal que el anterior, y desde cierto punto de vista puede ser más fácil de comprender. Sin embargo, su realización requiere de 8 a 20 términos AND por salida, en comparación a solamente 3 a 5 por salida para la versión original del contador en anillo. Éste es un buen ejemplo de cómo se puede mejorar la eficiencia y desempeño del circuito mediante la adaptación de una estructura estándar simple para un problema de diseño "particular", más que forzar una máquina de estado por fuerza bruta.

RESTABLECIMIENTO (RESET) CONFIABLE

Note en la tabla 8-27 que se asigna a TSTATE un valor en la sección equations del programa, aunque ya se había utilizado en la sección state_diagram. Esto se ha hecho con un propósito muy específico, para asegurar que el programa se dirija al estado SRESET desde cualquier estado indefinido, como se explica a continuación.

ABEL aumenta el conjunto de iniciación por una salida cada vez que la salida aparece en el lado izquierdo de una ecuación, como explicamos para las salidas combinacionales en la página 254. En el caso de salida registradas, ABEL también aumenta el conjunto de iniciación de cada variable de estado en el vector de estado para cada definición "state" en un state_diagram. Para cada salida estado-variable, todas las combinaciones de entrada que provocan que esa salida sea 1 en cada state son sumadas al conjunto de iniciación de la salida.

La máquina de estado en la tabla 8-27 tiene 2^7 o 128 estados en total, de los cuales solamente 13 están explícitamente definidos y tienen una transición en SRESET. No obstante, la ecuación WHEN asegura que en cualquier momento que RESET sea asertiva, la máquina vaya al estado SRESET. Esto es verdadero sin tener en cuenta las definiciones state en el state_diagram. Cuando RESET es asertiva, la codificación del estado con la totalidad de unos es, en efecto, operada con OR con el estado siguiente, si hay, especificado por el state_diagram. Este enfoque para un restablecimiento confiable no sería posible si SRESET estuviera codificado con puros ceros, por ejemplo.

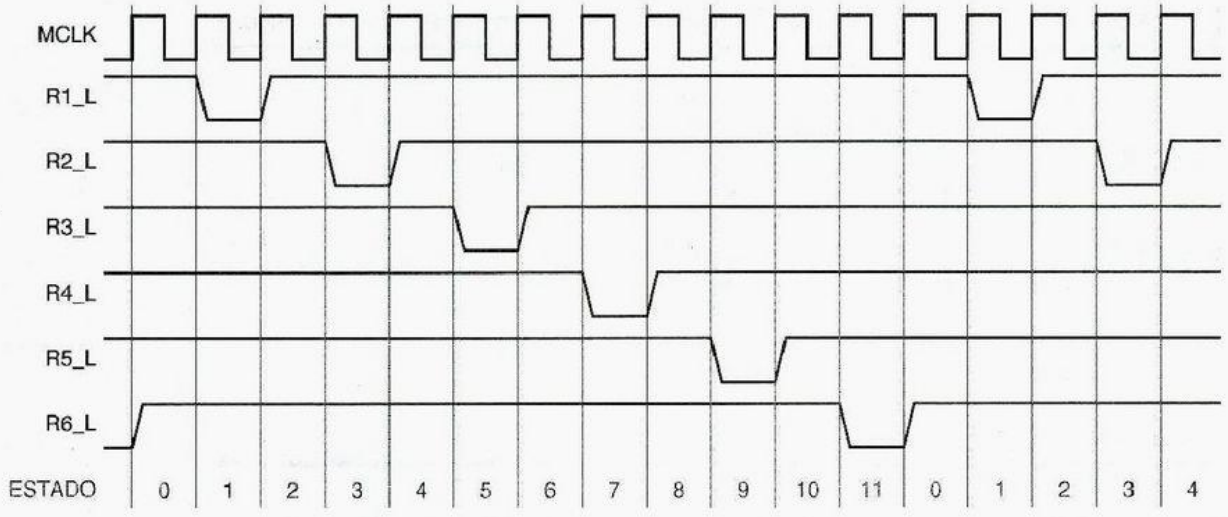


Figura 8-72 Formas de onda de temporización modificadas para un sistema digital.

Ahora examinemos una variación de las formas de onda previas de temporización que se pueden requerir en un sistema diferente. La figura 8-72 es semejante a las formas de onda anteriores, excepto que cada salida de fase Ri es asertiva por solamente un tic de reloj por fase. Este cambio tiene un efecto sutil pero importante en el enfoque del diseño.

En el diseño original, utilizamos un contador en anillo de 6 bits y un bit de estado auxiliar T1 para seguir la pista de los dos estados dentro de cada fase. Con las nuevas formas de onda esto no es posible. En los estados entre los pulsos de nivel activo bajo (STATE = 0, 2, 4, etc. en la figura 8-72) las salidas de fase son negadas todas, de modo que ya no se pueden utilizar para averiguar cuál estado debería visitarse a continuación. Es necesaria otra cosa para seguir la pista del estado.

Existen muchas maneras diferentes para resolver este problema. Una idea es empezar con el diseño original en la tabla 8-26 y utilizar las salidas de fase P1_L, P2_L y así sucesivamente solamente como estados internos. Entonces, cada salida de fase

Tabla 8-28 Adiciones a la tabla 8-26 para un generador de formas de onda de seis fases modificado.

```
module␣TIMEG12K␣
...
R1_L,␣R2_L,␣R3_L,␣R4_L,␣R5_L,␣R6_L␣␣␣␣␣␣␣␣␣␣pin␣istype␣'com';
...
OUTPUTS␣=␣[R1_L,␣R2_L,␣R3_L,␣R4_L,␣R5_L,␣R6_L];

equations
...
!OUTPUTS␣=␣!PHASES␣&␣!T1;

end␣TIMEG12K
```

Ri_L puede definirse como una salida combinacional tipo Moore que es asertiva cuando la correspondiente Pi_L es asertiva *y* estamos en el segundo tic de una fase. El código adicional ABEL para apoyar este primer enfoque se muestra en la tabla 8-28.

Este primer enfoque es fácil, y funciona bastante bien si las señales Pi_L van a ser utilizadas solamente como de habilitación o como otras entradas de control. Sin embargo, es una mala idea si estas señales van a ser empleadas como relojes, porque pueden tener interferencias, como explicaremos a continuación. Las señales Pi_L y T1 son todas las salidas de flip-flops cronometrados por el mismo reloj maestro MCLK. Aunque estas señales cambian aproximadamente al mismo tiempo, su temporización nunca es bastante exacta. Una salida puede cambiar más pronto que la otra; esto se conoce como *sesgo de temporización de salida.* Por ejemplo, supongamos que en la transición del estado 1 al 2 en la figura 8-71, P2_L se va al nivel BAJO antes de que T1 se vaya al nivel ALTO. En este caso, aparecerá una breve interferencia en la salida de R2_L.

sesgo de temporización de salida

Para obtener salidas libres de interferencias, deberíamos diseñar el circuito de manera que cada salida de fase sea una salida registrada. Una forma de hacer esto es construir un contador en anillo de 12 bits y emplear solamente salidas alternas para producir las formas de onda deseadas; un programa ABEL que utiliza este enfoque se muestra en la tabla 8-29.

Tabla 8-29 Programa ABEL para un generador de formas de onda de seis fases modificado.

```
module TIMEG12
title 'Modified six-phase Master Timing Generator'

" Input and Output pins
MCLK, RESET, RUN, RESTART                    pin;
P1_L, P2_L, P3_L, P4_L, P5_L, P6_L           pin istype 'reg';
P1A, P2A, P3A, P4A, P5A, P6A                 pin istype 'reg';

" State definitions
PHASES = [P1A, P1_L, P2A, P2_L, P3A, P3_L, P4A, P4_L, P5A, P5_L, P6A, P6_L];
NEXTPH = [P6_L, P1A, P1_L, P2A, P2_L, P3A, P3_L, P4A, P4_L, P5A, P5_L, P6A];
SRESET = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
P1 =     [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];

equations

PHASES.CLK = MCLK;

WHEN RESET THEN PHASES := SRESET;
ELSE WHEN RESTART # (PHASES == SRESET) THEN PHASES := P1;
ELSE WHEN RUN THEN PHASES := NEXTPH;
ELSE PHASES := PHASES;

end TIMEG12
```

Tabla 8-30 Programa basado en contador para un generador de formas de onda de seis fases.

```
module TIMEG12A
title 'Counter-based six-phase master timing generator'

" Input and Output pins
MCLK, RESET, RUN, RESTART                pin;
P1_L, P2_L, P3_L, P4_L, P5_L, P6_L       pin istype 'reg';
CNT3..CNT0                               pin istype 'reg';

" Definitions
CNT = [CNT3..CNT0];
P_L = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];

equations

CNT.CLK = MCLK;  P_L.CLK = MCLK;

WHEN RESET THEN CNT := 15
ELSE WHEN RESTART THEN CNT := 0
ELSE WHEN (RUN & (CNT < 11)) THEN CNT := CNT + 1
ELSE WHEN RUN THEN CNT := 0
ELSE CNT := CNT;

P1_L := !(CNT == 0);
P2_L := !(CNT == 2);
P3_L := !(CNT == 4);
P4_L := !(CNT == 6);
P5_L := !(CNT == 8);
P6_L := !(CNT == 10);

end TIMEG12A
```

Otro enfoque más es reconocer que, puesto que las formas de onda pasan a través de 12 estados, podemos construir un contador binario módulo 12 y decodificar los estados de ese contador. Un programa ABEL que utiliza este enfoque se muestra en la tabla 8-30. Los estados del contador corresponden a los valores "STATE" mostrados en la figura 8-72. Ya que las salidas de fase son registradas, están libres de interferencias. Advierta que son decodificadas un ciclo antes, para explicar el retardo de decodificación de un tic. También, durante el restablecimiento, el contador es forzado al estado 15 más que al 0, de modo que la salida P1_L no es asertiva durante el restablecimiento.

8.5.10 Registros de corrimiento en VHDL

Los registros de corrimiento se pueden especificar en forma estructural o de comportamiento en VHDL; examinaremos unas cuantas aplicaciones y descripciones de comportamiento. La tabla 8-31 es la tabla de función para un registro de corrimiento de 8 bits con un conjunto extendido de funciones. Además de las funciones de retención, carga y

Tabla 8-31 Tabla de función para un registro de corrimiento de 8 bits de función extendida.

	Entrada			Estado siguiente							
Función	*S2*	*S1*	*S0*	*Q7**	*Q6**	*Q5**	*Q4**	*Q3**	*Q2**	*Q1**	*Q0**
Retención	0	0	0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
Carga	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Corrimiento a la derecha	0	1	0	RIN	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Corrimiento a la izquierda	0	1	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	LIN
Corrimiento circular a la derecha	1	0	0	Q0	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Corrimiento circular a la izquierda	1	0	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	Q7
Corrimiento aritmético a la derecha	1	1	0	Q7	Q7	Q6	Q5	Q4	Q3	Q2	Q1
Corrimiento aritmético a la izquierda	1	1	1	Q6	Q5	Q4	Q3	Q2	Q1	Q0	0

corrimiento del 74x194 y 74x299, efectúan operaciones de conocimiento aritmético y circular. En las operaciones de *corrimiento circular*, el bit que "sale" por un extremo durante un corrimiento se retroalimenta en el otro extremo. En las operaciones de *corrimiento aritmético*, la entrada de frontera se establece para multiplicación o división por 2; para un corrimiento a la izquierda, la entrada derecha es 0, para un corrimiento a la derecha, el bit más a la izquierda (de signo) es reproducido.

corrimiento circular
corrimiento aritmético

Un programa de comportamiento VHDL para el registro de corrimiento de función extendida se muestra en la tabla 8-32. Como en ejemplos anteriores, definimos un proceso y empleamos el atributo `event` en la señal de `CLK` para obtener el comportamiento disparado por flanco deseado. Vale la pena advertir las otras características de este programa:

- Una señal interna, `IQ`, es utilizada para lo que finalmente llegue a ser la salida `Q`, de modo que puede ser tanto leída como escrita mediante proposiciones de proceso. De manera alternativa, podríamos haber definido la salida `Q` como de tipo "`buffer`".
- La entrada `CLR` es asincrónica debido a que se encuentra en la lista de sensibilidad del proceso, se prueba cada vez que cambia. Y la instrucción `IF` está estructurada de manera que `CLR` tenga procedencia sobre cualquier otra condición.
- Una instrucción `CASE` se utiliza para definir el funcionamiento de registro de corrimiento para los ocho posibles valores de las entradas de selección (`2 downto 0`).
- En la instrucción `CASE`, el caso "`when others`" es requerido para evitar que el compilador proteste sobre aproximadamente ¡2^{32} casos sin cubrir!
- La instrucción "`null`" indica que no se toma ninguna acción en ciertos casos. En el caso 1, advierta que no se requiere de una acción; la opción predeterminada es para que una señal mantenga su valor a menos que se establezca otra cosa.
- En la mayoría de los casos el operador de concatenación "`&`" se utiliza para construir un arreglo de 8 bits de un subconjunto de 7 bits de `IQ` y un bit aparte.

Tabla 8-32 Programa VHDL para un registro de corrimiento de 8 bits de función extendida.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vshftreg is
    port (
        CLK, CLR, RIN, LIN: in STD_LOGIC;
        S: in STD_LOGIC_VECTOR (2 downto 0); -- selección de la función
        D: in STD_LOGIC_VECTOR (7 downto 0); -- entrada de datos
        Q: out STD_LOGIC_VECTOR (7 downto 0) -- salida de datos
    );
end Vshftreg;

architecture Vshftreg_arch of Vshftreg is
signal IQ: STD_LOGIC_VECTOR (7 downto 0);
begin
process (CLK, CLR, IQ)
  begin
    if (CLR='1') then IQ <= (others=>'0'); -- borrado asíncrono
    elsif (CLK'event and CLK='1') then
      case CONV_INTEGER(S) is
        when 0 => null;                          -- Retención
        when 1 => IQ <= D;                       -- Carga
        when 2 => IQ <= RIN & IQ(7 downto 1);    -- Corrimiento a la derecha
        when 3 => IQ <= IQ(6 downto 0) & LIN;    -- Corrimiento a la izquierda
        when 4 => IQ <= IQ(0) & IQ(7 downto 1); -- Corrimiento circular a la derecha
        when 5 => IQ <= IQ(6 downto 0) & IQ(7); -- Corrimiento circular a la izquierda
        when 6 => IQ <= IQ(7) & IQ(7 downto 1); -- Corrimiento aritmético a la derecha
        when 7 => IQ <= IQ(6 downto 0) & '0';    -- Corrimiento aritmético a la izquierda
        when others => null;
      end case;
    end if;
    Q <= IQ;
  end process;
end Vshftreg_arch;
```

- Debido a los fuertes requerimientos de VHDL para la coincidencia de tipos, usaremos la función `CONV_INTEGER` que definimos en la sección 4.7.4 para convertir la entrada de selección `S` de `STD_LOGIC_VECTOR` a un entero en la instrucción `CASE`. De manera alternativa, podríamos haber escrito cada etiqueta de caso como un `STD_LOGIC_VECTOR` (por ejemplo, `('0','1','1')` en lugar del entero 3).

Una aplicación de los registros de corrimiento se encuentra en los contadores en anillo, como en el generador de formas de onda de seis fases que describimos en la página 736 con las formas de onda en la figura 8-71. Un programa VHDL que proporciona el comportamiento correspondiente se muestra en la tabla 8-33. Como en el anterior ejem-

Tabla 8-33 Programa VHDL para un generador de formas de onda de seis fases.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vtimegn6 is
    port (
        MCLK, RESET, RUN, RESTART: in STD_LOGIC; -- entradas de reloj, control
        P_L: out STD_LOGIC_VECTOR (1 to 6)       -- salidas de fase de nivel activo bajo
    );
end Vtimegn6;

architecture Vtimegn6_arch of Vtimegn6 is
signal IP: STD_LOGIC_VECTOR (1 to 6); -- señales de fase de nivel activo alto internas
signal T1: STD_LOGIC;                 -- primer tic dentro de la fase
begin
process (MCLK, IP)
  begin
    if (MCLK'event and MCLK='1') then
      if (RESET='1') then
        T1 <= '1'; IP <= ('0','0','0','0','0','0');
      elsif ((IP=('0','0','0','0','0','0')) or (RESTART='1')) then
        T1 <= '1'; IP <= ('1','0','0','0','0','0');
      elsif (RUN='1') then
        T1 <= not T1;
        if (T1='0') then IP <= IP(6) & IP(1 to 5); end if;
      end if;
    end if;
    P_L <= not IP;
  end process;
end Vtimegn6_arch;
```

plo de VHDL, un vector de señal de nivel activo alto interno, `IP`, se utiliza para leer y escribir lo que finalmente llega a ser la salida del circuito; esta señal interna es convenientemente invertida en la última instrucción para obtener el vector de señal de salida de nivel activo bajo requerido. El resto del programa es directo, pero advierta que tiene tres niveles de proposiciones `IF` anidadas.

Una posible modificación a la aplicación anterior es producir formas de onda de salida que sean asertivas solamente durante el segundo tic de cada fase de dos tics; tales formas de onda se mostraron en la figura 8-72. Una manera de hacer esto es crear un contador en anillo de 12 bits y utilizar solamente salidas alternas. En la realización de VHDL solamente las salidas de seis fases, `P_L(1 to 6)`, aparecerían en la definición de entidad. Las seis señales adicionales, que nombramos `NEXTP(1 to 6)`, son locales a la definición de la arquitectura. La figura 8-73 muestra la relación de estas señales para operación de registro de corrimiento, y la tabla 8-34 es el programa VHDL.

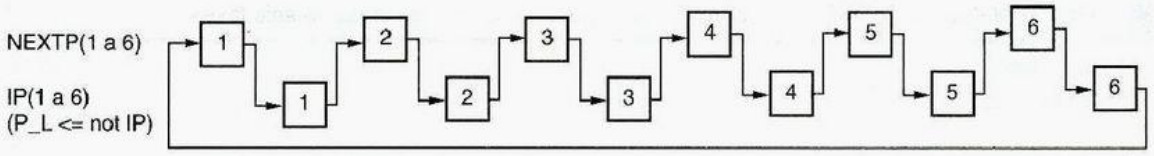


Figura 8-73 Secuencia de corrimiento para el contador en anillo de 12 bits generador de formas de onda.

Como en el programa anterior, una señal de nivel activo alto de 6 bits, IP, es declarada en el cuerpo de la arquitectura y utilizada para leer y escribir lo que finalmente llega a ser la salida de nivel activo bajo del circuito, P_L. La señal de 6 bits adicional, NEXTP, retiene los 6 bits restantes de estado. Las constantes IDLE y FIRSTP se emplean para mejorar la legibilidad del programa.

Tabla 8-34 Programa VHDL para un generador de formas de onda de seis fases modificado.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vtimeg12 is
    port (
        MCLK, RESET, RUN, RESTART: in STD_LOGIC; -- entradas de reloj, control
        P_L: out STD_LOGIC_VECTOR (1 to 6)       -- salidas de fase de nivel activo bajo
    );
end Vtimeg12;

architecture Vtimeg12_arch of Vtimeg12 is
signal IP, NEXTP: STD_LOGIC_VECTOR (1 to 6); -- señales de fase de nivel activo alto internas
begin
process (MCLK, IP, NEXTP)
  variable TEMP: STD_LOGIC_VECTOR (1 to 6);  -- temporal para corrimiento de señal
  constant IDLE: STD_LOGIC_VECTOR (1 to 6) := ('0','0','0','0','0','0');
  constant FIRSTP: STD_LOGIC_VECTOR (1 to 6) := ('1','0','0','0','0','0');
  begin
    if (MCLK'event and MCLK='1') then
      if (RESET='1') then IP <= IDLE;  NEXTP <= IDLE;
      elsif (RESTART='1') or (IP=IDLE and NEXTP=IDLE) then IP <= IDLE;  NEXTP <= FIRSTP;
      elsif (RUN='1') then
        if (IP=IDLE) and (NEXTP=IDLE) then NEXTP <= FIRSTP;
        else TEMP := IP;  IP <= NEXTP;  NEXTP <= TEMP(6) & TEMP(1 to 5);
        end if;
      end if;
    end if;
    P_L <= not IP;
  end process;
end Vtimeg12_arch;
```

Note que una variable de 6 bits, TEMP, se utiliza sólo como un lugar temporal para retener el valor antiguo de IP cuando se presenta el corrimiento, IP se carga con NEXTP, y NEXTP es cargada con el valor antiguo, desplazado de IP. Debido a que las proposiciones de asignación en un proceso se ejecutan *secuencialmente*, no podríamos escaparnos solamente escribiendo "IP <= NEXTP; NEXTP <= IP(6) & IP(1 to 5);";. Si hiciéramos eso, entonces NEXTP seleccionaría el *nuevo* valor de IP, no el antiguo. Note también que puesto que TEMP es una variable local, no una señal, su valor no es visible fuera del proceso. También, una invocación de proceso nunca emplea el valor de TEMP de una invocación anterior. Por lo tanto, el compilador VHDL no necesita sintetizar ningún flip-flop para retener el valor de TEMP.

*8.6 Circuitos iterativos contra secuenciales

Presentamos los circuitos iteractivos en la sección 5.9.2. La función de un circuito iterativo módulo n puede efectuarse mediante un circuito secuencial que utilice sólo una copia del módulo; sin embargo, requiere de n pasos (tics de reloj) para obtener el resultado. Éste es un ejemplo excelente de un compromiso espacio/temporal en diseño digital.

Como se muestra en la figura 8-74, los flip-flops se utilizan en la versión de circuito secuencial para almacenar las salidas en cascada al final de cada paso; las salidas del flip-flop se emplean como las entradas en cascada al principio del paso siguiente. Los flip-flops deben ser inicializados a los valores de entrada a la frontera antes del primer tic de reloj, y contienen los valores de salida a la frontera después del n ésimo tic.

Puesto que un circuito iterativo es un circuito combinacional, todas sus entradas primarias y de frontera pueden aplicarse simultáneamente y sus salidas primarias y de frontera están todas disponibles después de un retardo combinacional. En la versión de circuito secuencial, las entradas primarias deben ser entregadas de manera secuencial, una por cada tic de reloj, y las salidas primarias son producidas con temporización similar. Por lo tanto, los registros de corrimiento de salida serie se emplean con frecuencia para proporcionar las entradas, y los registros de corrimiento de entrada serie se utilizan para recolectar las salidas. Por esta razón, la versión de circuito secuencial de un "artefacto iterativo" se denomina con frecuencia un "artefacto serie".

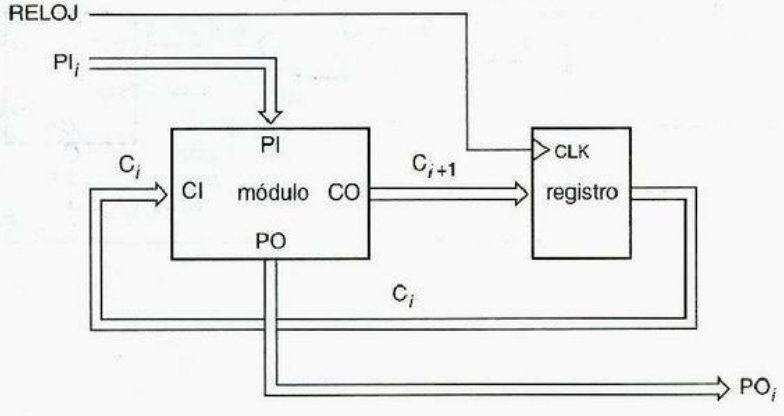


Figura 8-74
Estructura general de la versión de circuito secuencial de un circuito iterativo.

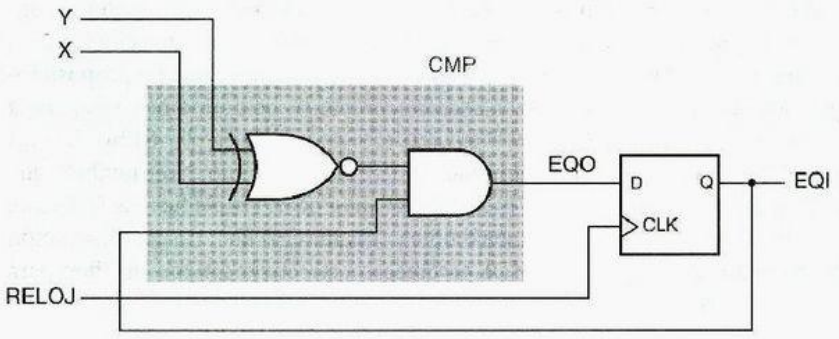


Figura 8-75
Circuito comparador serie simplificado.

comparador serie

Por ejemplo, la figura 8-75 muestra el diseño básico para un circuito *comparador serie*. El bloque sombreado es idéntico al módulo que se utiliza en el comparador iterativo de la figura 5-80. El circuito se dibuja con más detalle utilizando los chips SSI de la figura 8-76. Además, hemos proporcionado una entrada de restablecimiento sincrónica que, cuando es asertiva, obliga al valor inicial del flip-flop en cascada a 1 en el siguiente tic de reloj. El valor inicial del flip-flop en cascada corresponde a la entrada de frontera en el comparador iterativo.

Con el comparador serie, una comparación de n bits requiere de $n + 1$ tics de reloj. RESET_L es asertiva en el primer tic de reloj. RESET_L es negada y se aplican los bits de datos en los siguientes n tics. La salida EQI proporciona el resultado de comparación durante el periodo de reloj que sigue al último tic. Un diagrama de temporización para dos comparaciones sucesivas de 4 bits se muestra en la figura 8-77. Los picos en la forma de onda EQO indican el tiempo en que las salidas combinacionales se establecen en respuesta a los nuevos valores X y Y de entrada.

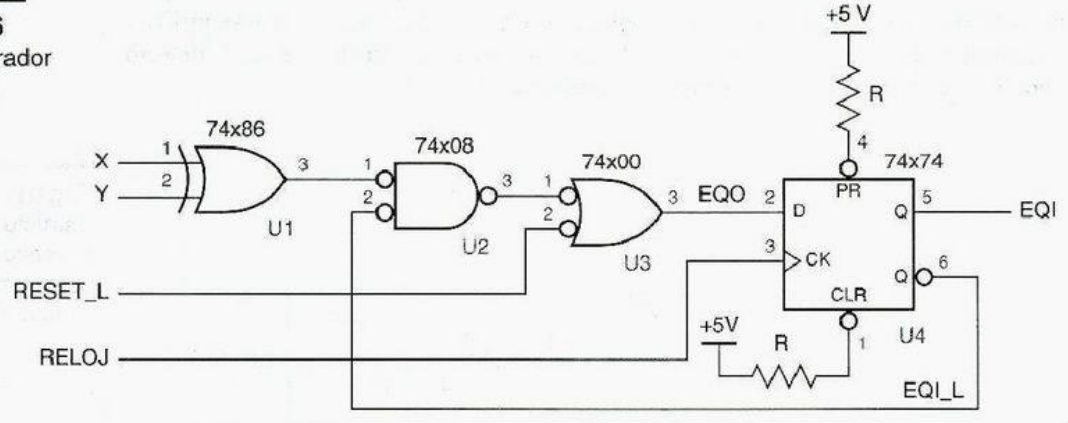


Figura 8-76
Circuito comparador serie detallado.

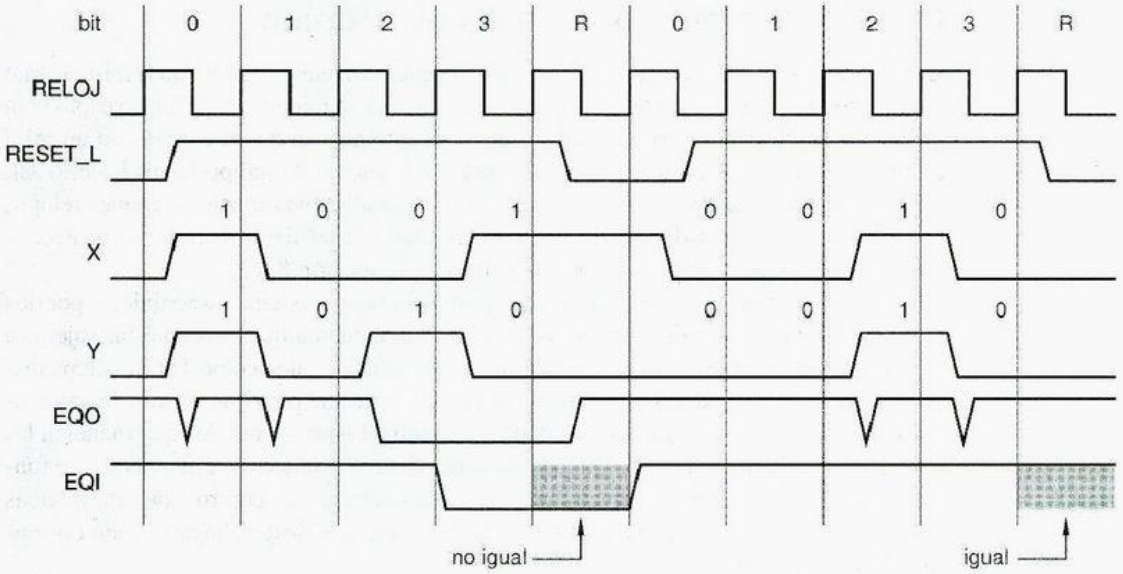


Figura 8-77 Diagrama de temporización para circuito comparador serie.

sumador binario en serie

Un circuito *sumador binario serie* para sumandos de cualquier longitud se pueden construir a partir de un sumador completo y un flip-flop D, como se muestra en la figura 8-78. El flip-flop, que almacena el acarreo entre bits sucesivos de la suma, se borra a 0 en el restablecimiento. Los bits de sumando se presentan en serie en las entradas A y B, comenzando con el LSB, y los bits de suma aparecen en S en el mismo orden.

Debido al gran tamaño y alto costo de los circuitos de lógica digital en los primeros días de esta tecnología, muchas computadoras y calculadoras utilizaban sumadores en serie y otras versiones en serie de circuitos iteractivos para efectuar operaciones aritméticas. Aun cuando estos circuitos aritméticos no se emplean mucho hoy en día, es un recordatorio formativo de los compromisos espacio/temporales que son posibles en el diseño digital.

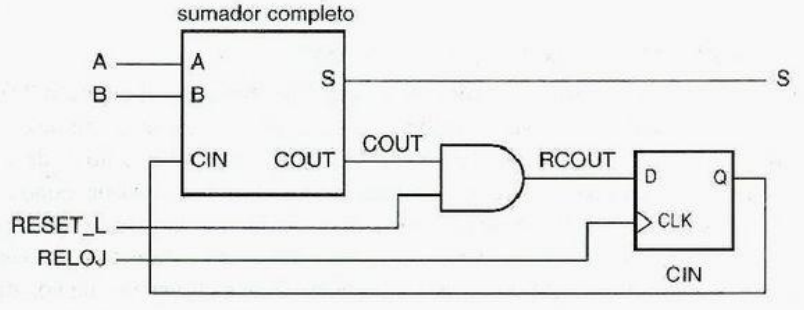


Figura 8-78
Circuito sumador binario serie.

8.7 Metodología de diseño sincrónico

sistema síncrono

En un *sistema sincrónico*, todos los flip-flops se encuentran temporizados con la misma señal de reloj común, y no se emplean las entradas de prestablecimiento y borrado, excepto para la inicialización del sistema. Aunque todo el mundo no marcha al compás de un reloj común, dentro de los confines de un sistema o subsistema digital podemos hacerlo así. Cuando interconectamos sistemas o subsistemas digitales que utilizan diferentes relojes, por lo regular podemos identificar un número limitado de señales asincrónicas que necesitan de un tratamiento especial, como veremos en la sección 8.8.3.

Las carreras y los riesgos no son un problema en los sistemas sincrónicos, por dos razones. Primero, los únicos circuitos en modo fundamental que pueden estar sujetos a carreras o riesgos esenciales son elementos prediseñados, tales como los flip-flops discretos o celdas ASIC, que son garantizados por el fabricante para que funcionen apropiadamente. En segundo lugar, aun cuando los circuitos combinacionales que manejan las entradas de control de flip-flop pueden contener riesgos estáticos, o dinámicos o de función, estos riesgos no tienen efecto, puesto que las entradas de control son muestreadas solamente *después* de que las interferencias inductoras de riesgos hayan tenido oportunidad de establecerse.

Aparte de diseñar el comportamiento funcional de cada máquina de estado, el diseñador de un sistema o subsistema sincrónico práctico debe efectuar justamente tres tareas bien definidas para asegurar un funcionamiento confiable del sistema:

1. Minimizar y determinar la magnitud de sesgo de reloj en el sistema, como se discutió en la sección 8.8.1.
2. Asegurar que los flip-flops tengan márgenes positivos de tiempo de establecimiento y de retención, incluyendo una asignación para el sesgo de reloj, como se describió en la sección 8.1.4.
3. Identificar las entradas asincrónicas, sincronizándolas con el reloj y asegurar que los sincronizadores tengan una probabilidad adecuadamente baja de falla, como se describe en las secciones 8.8.3 y 8.9.

Antes de entrar en estas cuestiones, en esta sección examinaremos un modelo general para la estructura del sistema sincrónico y también un ejemplo.

8.7.1 Estructura del sistema síncrono

Los ejemplos del diseño de circuito secuencial que dimos en el capítulo 7 fueron en su mayor parte máquinas de estado individuales con un pequeño número de estados. Si un circuito secuencial tiene más de unos cuantos flip-flops, entonces no es deseable (y frecuentemente ni es posible) tratar el circuito como una máquina de estado monolítica simple, porque el número de estados sería demasiado grande para ser manejable.

Afortunadamente, la mayoría de los sistemas o subsistemas digitales se pueden dividir en dos o más partes. Si el sistema procesa números, señales de voz digitalizadas o un flujo de pulsos de una bujía, cierta parte del sistema, que llamamos la *unidad de datos*, puede ser vista como la que almacena, dirige, combina y generalmente procesa los "datos". Otra parte, que denominamos la *unidad de control*, puede verse como la que inicia y detiene las acciones en la unidad de datos, prueba las condiciones y decide

unidad de datos
unidad de control

lo que hay que hacer a continuación de acuerdo a las circunstancias. En general, solamente la unidad de control debe ser diseñada como una máquina de estado. La unidad de datos y sus componentes habitualmente se maneja a un nivel más alto de abstracción, tal como:

- *Registros*. Una colección de flip-flops es cargada en paralelo con muchos bits de "datos", los cuales pueden entonces ser usados o recuperados en conjunto.
- *Funciones especializadas*. Éstas incluyen de corrimiento y contadores de bits múltiples, los cuales incrementan o desplazan su contenido bajo una orden.
- *Memoria de lectura/escritura*. Los latchs o flip-flops individuales en un conjunto de los mismos pueden escribirse o leerse.

Los primeros dos puntos anteriores fueron discutidos al principio de este capítulo y el último se discute en el capítulo 10.

La figura 8-79 es un diagrama de bloques general de un sistema con una unidad de control y una unidad de datos. También hemos incluido bloques explícitos para la entrada y salida, pero podríamos tan sólo haber absorbido fácilmente estas funciones en la misma unidad de datos. La unidad de control es una máquina de estado cuyas entradas incluyen *entradas de mando* que indican cómo la máquina va a funcionar, y *entradas de condición* proporcionadas por la unidad de datos. Las entradas de mando se pueden suministrar mediante otro subsistema o usuario para establecer el modo de operación general de la máquina de estado de control (RUN/HALT, NORMAL/TURBO, etc.), mientras las entradas de condición permiten a la "máquina de estado de la unidad de control" cambiar su comportamiento como sea requerido por las circunstancias en la unidad de datos (ZERO_DETECT, MEMORY_FULL, etcétera).

entrada de mando
entrada de condición

Una característica clave de la estructura en la figura 8-79 es que las unidades de control, datos, entrada y salida utilizan todas el mismo reloj común. La figura 8-80 ilustra las operaciones de las unidades de control y de datos durante un ciclo de reloj típico:

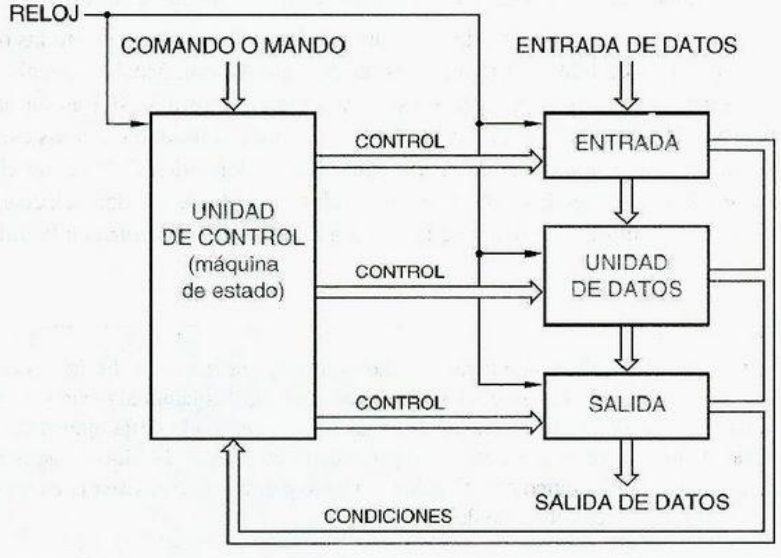


Figura 8-79
Estructura de sistema síncrónico.

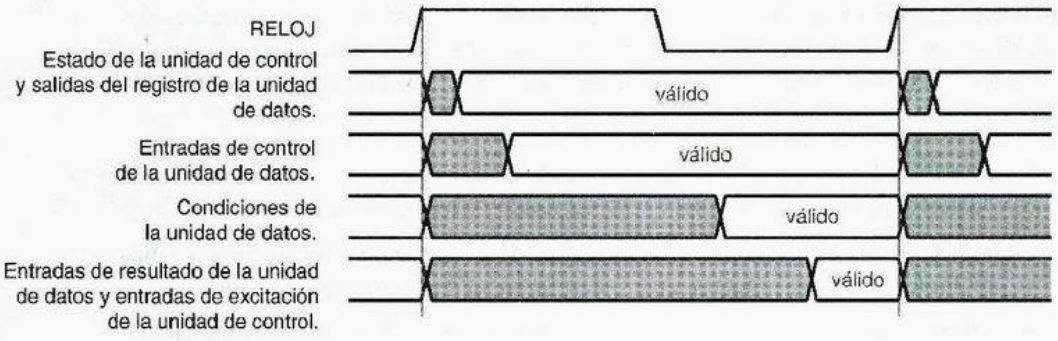


Figura 8-80 Operaciones durante un ciclo de reloj en un sistema sincrónico.

1. Poco después del inicio del periodo del reloj, el estado de la unidad de control y las salidas del registro de la unidad de datos son válidas.
2. Acto seguido, después de un retardo lógico combinacional, las salidas tipo Moore de la máquina de estado de la unidad de control llegan a ser válidas. Estas señales son *entradas* de control para la unidad de datos. Determinan qué funciones de la unidad de datos son realizadas en el resto del periodo de reloj: por ejemplo, selección de las direcciones de memoria, trayectorias del multiplexor y operaciones aritméticas.
3. Cerca del final del periodo de reloj, las salidas de condición de la unidad de datos tales como las de detección de cero o de sobrecarga son válidas y se quedan disponibles para la unidad de control.
4. Al final del periodo de reloj, justamente antes de que comience la ventana de establecimiento de tiempo, la lógica del estado siguiente de la máquina de estado de la unidad de control ha determinado el estado siguiente basada en el estado actual y las entradas de mando y de condición. Aproximadamente al mismo tiempo, los resultados de cálculo en la unidad de datos están disponibles para ser cargados en los registros de la unidad de datos.
5. Después del flanco de reloj, el ciclo completo puede repetirse.

Las entradas de control de la unidad de datos, las cuales son salidas de máquina de estado de la unidad de control, pueden ser del tipo Moore, Mealy o Mealy segmentadas; la temporización para el tipo Moore se muestra en la figura 8-80. Las salidas tipo Moore y tipo Mealy segmentadas controlan las acciones de la unidad de datos estrictamente de acuerdo al estado actual y entradas pasadas, que no dependen de las condiciones *actuales* en la unidad de datos. En contraste, las salidas tipo Mealy pueden seleccionar diferentes acciones en la unidad de datos de acuerdo a condiciones *actuales* en la unidad de datos.

SALIDAS DE MEALY SEGMENTADAS

Algunas máquinas de estado tienen salidas de Mealy segmentadas discutidas en la sección 7.3.2. En la figura 8-80, las salidas de Mealy segmentadas habitualmente serían válidas al principio del ciclo, al mismo tiempo que las salidas de estado de la unidad de control. La validez temprana de estas salidas, comparada con las salidas de Moore que deben pasar a través de un retardo lógico combinacional, puede permitir que el sistema entero funcione a una velocidad de reloj más rápida.

Esto incrementa la flexibilidad, pero habitualmente también incrementa el periodo de reloj mínimo para corregir la operación del sistema, puesto que la trayectoria de retardo puede ser mucho más extensa. También, las salidas tipo Mealy no deben crear lazos de retroalimentación. Por ejemplo, una señal que agrega 1 a una entrada de sumador si la salida del sumador es distinta de cero ocasiona una oscilación si la salida del sumador es –1.

8.7.2 Un ejemplo de diseño de sistema sincrónico

Para darle a usted una perspectiva general de los diversos elementos del diseño de sistemas sincrónicos, esta subsección presenta un ejemplo representativo de un sistema sincrónico. El ejemplo es un *multiplicador de corrimiento y suma* para enteros sin signo que utiliza el algoritmo de la sección 2.8. Su unidad de datos utiliza bloques de construcción secuencial y combinacional estándar, y su unidad de control es descrita por un diagrama de estado.

multiplicador de corrimiento y suma

La figura 8-8 ilustra funciones y registros de unidad de datos que se emplean para efectuar una multiplicación de 8 bits:

MPY/LPROD Un registro de corrimiento que inicialmente almacena el multiplicador, y acumula los bits de menor orden del producto a medida que se ejecuta el algoritmo.

HPROD Un registro que inicialmente está borrado, y acumula los bits de orden mayor del producto a medida que se ejecuta el algoritmo.

MCND Un registro que almacena el multiplicando a lo largo del algoritmo.

F Una función combinacional igual a la suma de 9 bits de HPROD y MCND si el bit de menor orden de MPY/LPROD es 1, y es igual a HPROD (extendido a 9 bits) de otro modo.

El registro de corrimiento MPY/LPROD sirve para un doble propósito, manteniendo tanto los bits del multiplicador que faltan de verificar (a la derecha) como los bits de producto sin cambiar (a la izquierda) a medida que el algoritmo se ejecuta. En cada paso se corre a la derecha un bit, descartando el bit multiplicador que se acaba de probar, moviendo el siguiente bit multiplicador por probarse a la posición más a la derecha, y cargando en la posición más a la izquierda un bit de producto más que no cambiará por el resto del algoritmo.

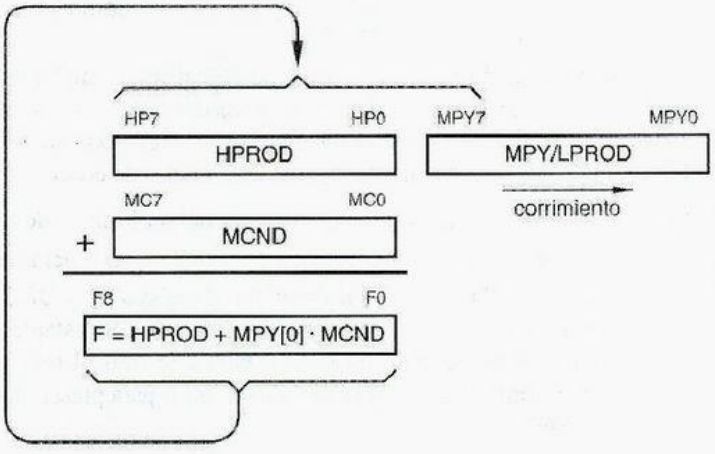


Figura 8-81
Registros y funciones utilizadas por el algoritmo de multiplicación de corrimiento y suma.

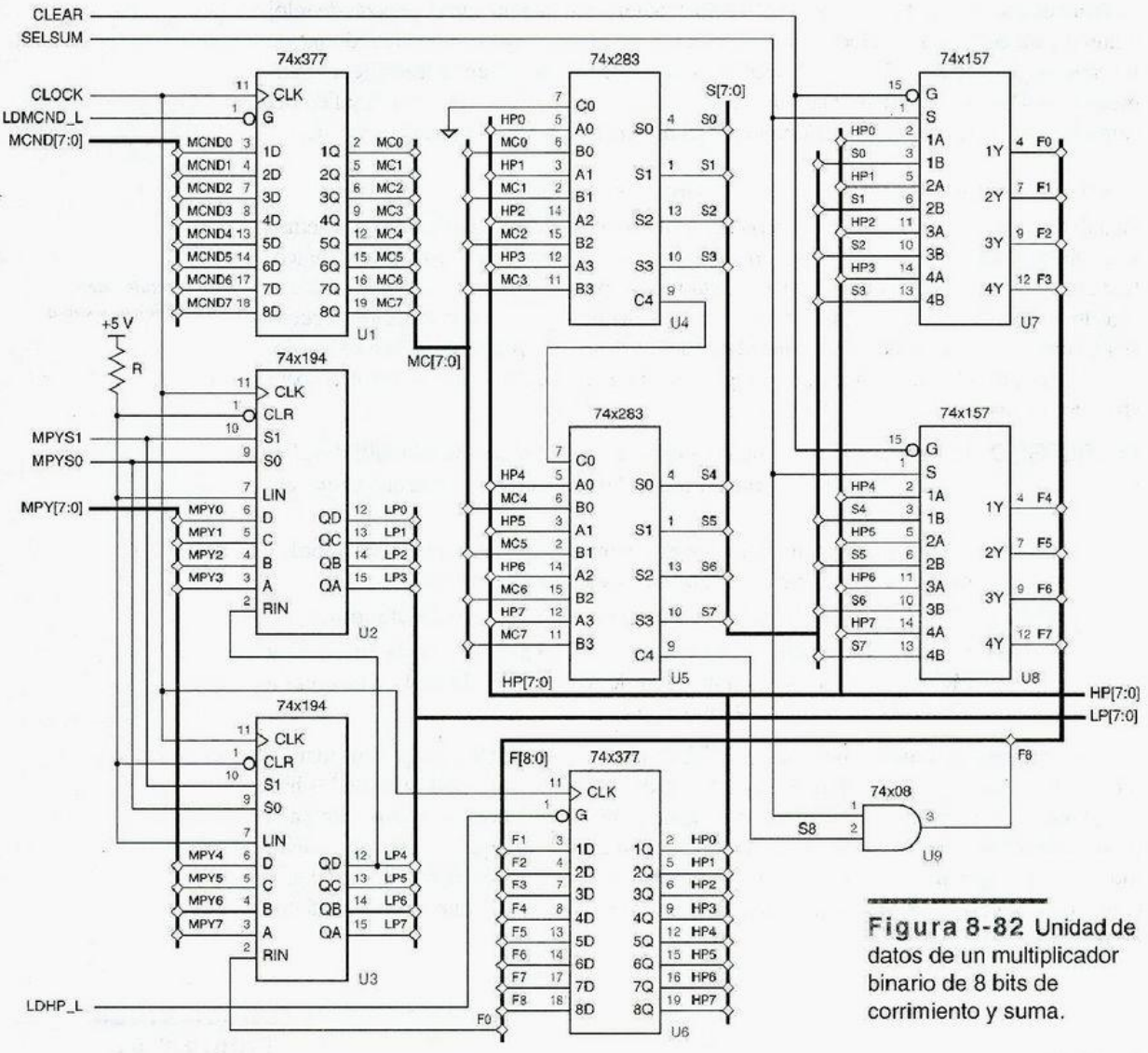


Figura 8-82 Unidad de datos de un multiplicador binario de 8 bits de corrimiento y suma.

La figura 8-82 es un diseño MSI para la unidad de datos. El multiplicador, MPY[7:0], así como el multiplicando, MCND[7:0], se cargan en dos registros antes de que comience la multiplicación. Cuando la multiplicación está completa, el producto aparece en HP[7:0] y LP[7:0]. La unidad de datos utiliza las siguientes señales de control:

LDMCND_L Su puesta a 1, habilita el registro U1 del multiplicando para ser cargado.

LDHP_L Su puesta a 1, habilita el registro U6 de HPROD para ser cargado.

MPYS[1:0] Cuando es 11, estas señales habilitan el registro U2 y U3 de MPY/LPROD para ser cargados en el siguiente tic de reloj. Se establecen a 01 durante la operación de multiplicación para habilitar el registro a la derecha del corrimiento, y a 00 en otros momentos para preservar el contenido del registro.

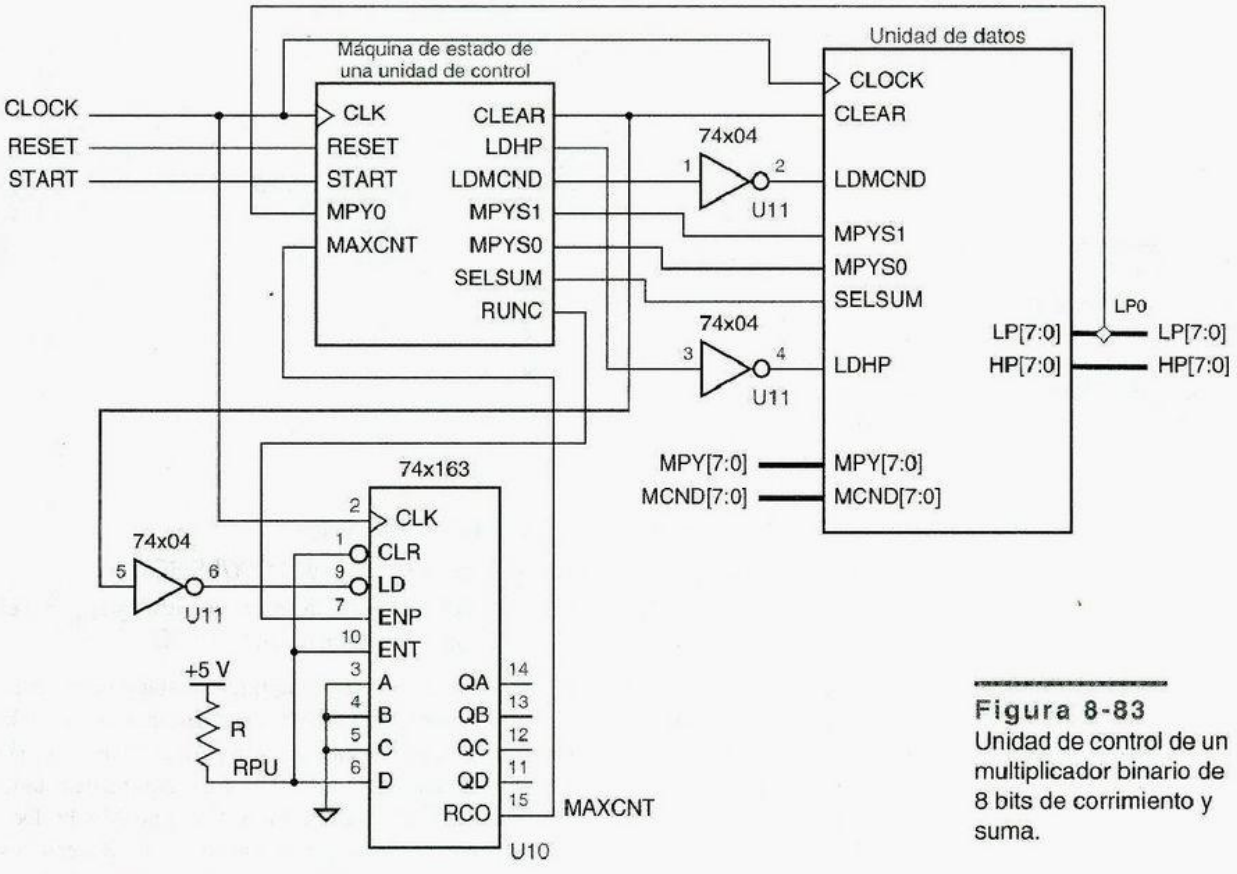


Figura 8-83
Unidad de control de un multiplicador binario de 8 bits de corrimiento y suma.

SELSUM Cuando ésta se afirma, los multiplexores U7 y U8 seleccionan la salida de los sumadores U4 y U5, que es la suma de HPROD y el multiplicando MC. De otro modo, seleccionan directamente HPROD.

CLEAR Cuando se afirma, la salida de los multiplexores U7 y U8 es cero.

El multiplicador utiliza una unidad de control, que se muestra junto con el bloque de la unidad de datos en la figura 8-83, para inicializar la unidad de datos y pasar a través de una multiplicación. La unidad de control se descompone en un contador (U10) y una máquina de estado con el diagrama de estado mostrado en la figura 8-84.

La máquina de estado tiene las entradas y salidas siguientes:

RESET Una entrada de restablecimiento (reset) que es asértiva en el encendido.

START Una entrada de mando externa que inicia una multiplicación.

MPY0 Un entrada de condición desde la unidad de datos, el siguiente bit multiplicador que se va a probar.

CLEAR Una salida de control que hace cero la salida del multiplicador e inicializa el contador.

LDMCND Una salida de control que habilita el registro MCND para ser cargado.

LDHP Una salida de control que habilita el registro HPROD para ser cargado.

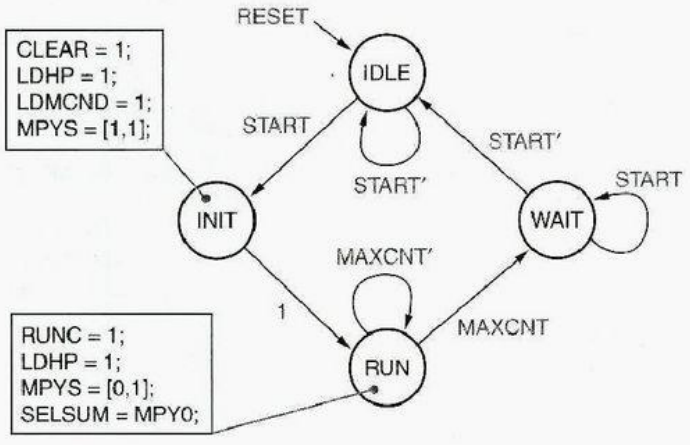


Figura 8-84
Diagrama de estado para la máquina de estado de control para un multiplicador binario de corrimiento y suma.

RUNC Una salida de control que habilita al contador.

MPYS[1:0] Salidas de control para corrimiento y carga de MPY/LPROD.

SELSUM Una salida de control que selecciona entre la salida corrida agregada o el HPROD corrido para ser cargado nuevamente en HPROD.

El diagrama de estado se puede transformar en una máquina de estado correspondiente utilizando una amplia variedad de métodos, que van desde el diseño basado en la realización de cálculos (es decir, el trabajo a mano) hasta el diseño para síntesis automática que utiliza la descripción correspondiente en ABEL o VHDL. La máquina de estado tiene en su mayor parte salidas tipo Moore; SELSUM es una salida tipo Mealy. Dos cuadros en el diagrama de estado enumeran las salidas que son asertivas en los estados INIT y RUN; todas las salidas son negadas en otros instantes. La máquina es diseñada de modo que al afirmar RESET en cualquier estado la lleve al estado IDLE.

Después de que la señal START es asertiva, comienza una multiplicación en el estado INIT. En este estado, el contador se inicializa a 1000_2, el multiplicador y el multiplicando se cargan en sus respectivos registros, y HPROD es borrado. El estado RUN se accesa enseguida y el contador se habilita para contar. La máquina de estado permanece en el estado RUN durante ocho tics de reloj, para ejecutar los ocho pasos del algoritmo de corrimiento y suma de 8 bits. Durante el octavo tic, el contador se encuentra en el estado 1111_2, de modo que MAXCNT es asertiva y la máquina de estado se dirige al estado WAIT. La máquina espera aquí hasta que START es negada, para evitar que reinicie una multiplicación hasta que se afirme una vez más.

Los detalles de diseño de las unidades de datos y control son interesantes, pero el aspecto más importante que se observa en este ejemplo es que todos los elementos del circuito secuencial tanto para los datos como para el control son flip-flops disparados por flanco temporizados por la misma señal CLOCK común. De este modo, su temporización es consistente con el modelo en la figura 8-80 y el diseñador no necesita preocuparse de carreras, riesgos y operaciones asincrónicas. A menos que la realización de la máquina de estado sea muy lenta, la velocidad de reloj máxima del circuito total estará limitada principalmente por los retardos de propagación a través de la unidad de datos.

8.8 Impedimentos para el diseño sincrónico

Aunque el enfoque sincrónico es el método más directo y confiable el diseño de sistema digital, unas cuantas desagradables realidades pueden interponerse en el camino. Discutiremos éstas en la presente sección.

8.8.1 Sesgo de reloj

Los sistemas sincrónicos que utilizan flip-flops disparados por frontera funcionan de manera apropiada solamente si todos los flip-flops ven la frontera de reloj de disparo al mismo tiempo. La figura 8-85 muestra qué es lo que puede ocurrir en caso contrario. Aquí, dos flip-flops se encuentran teóricamente temporizados por la misma señal, pero la señal de reloj vista por FF2 está retardada por una cantidad significativa con respecto al reloj de FF1. Esta diferencia entre los tiempos de llegada de reloj en diferentes dispositivos se conoce como *sesgo de reloj* o *asincronía de reloj*.

sesgo de reloj

Hemos nombrado el reloj retardado de la figura 8-85(a) "CLOCKD". Si el retardo de propagación de FF1 desde CLOCK hasta Q1 es breve, y si la conexión física de Q1 hacia FF2 es corta, entonces el cambio en Q1 provocado por una frontera de CLOCK puede en realidad alcanzar a FF2 *antes* de la frontera correspondiente CLOCKD. En este caso, FF2 puede ir a un estado siguiente incorrecto determinado por el *siguiente* estado en FF1 en lugar del estado actual, como se ilustra en (b). Si el cambio en Q1 llega a FF2 sólo ligeramente antes con relación a CLOCKD, entonces la especificación de tiempo de retención de FF2 puede ser violada, en cuyo caso FF2 puede llegar a ser metaestable y producir una salida impredecible.

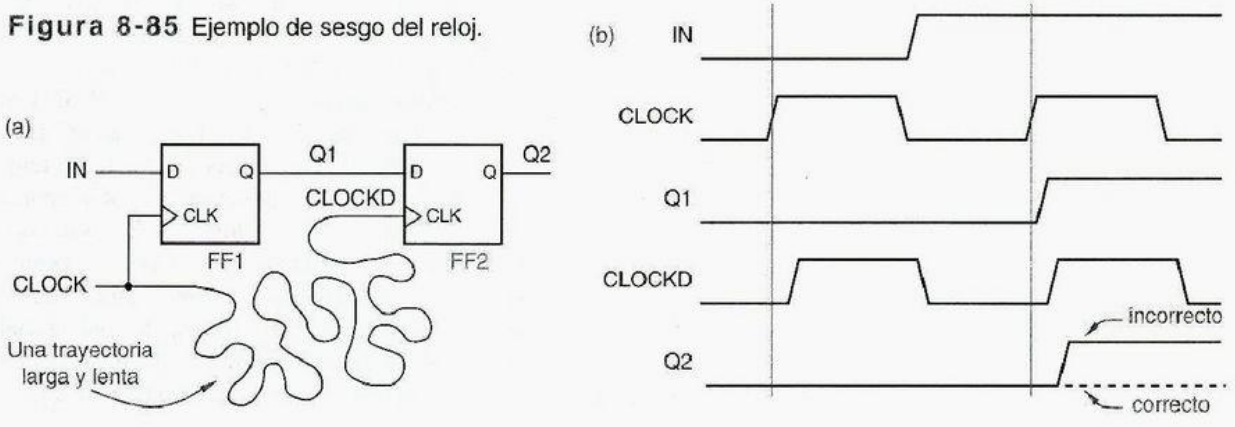
Si la figura 8-85 le recuerda el riesgo esencial mostrado en la figura 7-101, no anda mal. El problema del sesgo de reloj puede ser visualizado simplemente como una manifestación de los riesgos esenciales que existen en todos los dispositivos disparados por flanco.

Podemos determinar cualitativamente si el sesgo de reloj es un problema en un sistema determinado, definiendo t_{sesgo} como la cantidad de sesgo de reloj, y utilizamos los otros parámetros de temporización que se definen en la figura 8-1. Para un funcionamiento adecuado, necesitamos

$$t_{ffpd(min)} + t_{comb(min)} - t_{retención} - t_{sesgo(max)} > 0$$

En otras palabras, el sesgo de reloj se resta del margen de tiempo de retención que definimos en la sección 8.1.4.

Figura 8-85 Ejemplo de sesgo del reloj.



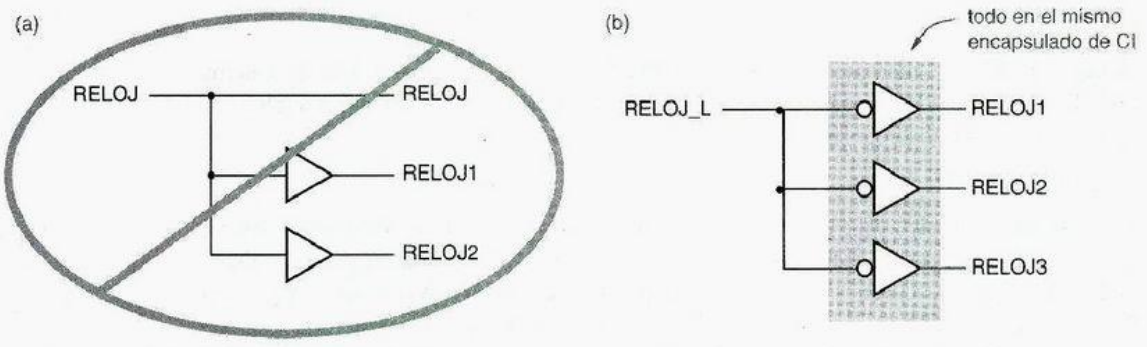


Figura 8-86 Retraso del reloj: (a) sesgo de reloj excesivo; (b) sesgo de reloj controlable.

Visto de manera aislada, el ejemplo de la figura 8-85 puede parecer un poco extremoso. Después de todo, ¿por qué proporcionaría un diseñador una trayectoria de conexión corta para los datos y una larga para el reloj, cuando simplemente podrían correr lado a lado? Existen varias maneras en que esto puede pasar; algunas son errores, mientras que otras son inevitables.

En un sistema grande, una sola señal de reloj puede no tener un fan out adecuado para controlar todos los dispositivos con entradas de reloj, de modo que es necesario proporcionar dos o más copias de la señal de reloj. El método de separación (buffering) de la figura 8-86(a) produce obviamente un sesgo de reloj excesivo, puesto que CLOCK1 y CLOCK2 son retardados a través de un buffer adicional comparado con CLOCK.

Un método recomendado se muestra en la figura 8-86(b). Todas las señales de reloj pasan a través de buffers idénticos, y así tienen retardos aproximadamente iguales. De manera ideal, todos los buffers deberían ser parte del mismo encapsulado de CI, de manera que todos tengan características de retardo similares y se encuentren funcionando a idéntica temperatura y voltaje de alimentación. Algunos fabricantes construyen buffers especiales para esta clase de aplicación y especifican la variación de retardo en el peor de los casos entre los buffers que están en el mismo encapsulado, lo que puede ser tan ínfimo como de unas cuantas décimas de nanosegundo.

Incluso el método de la figura 8-86(b) puede producir sesgo de reloj excesivo si una señal de reloj está mucho más pesadamente cargada que la otra; las transiciones en el reloj con más peso parecen ocurrir más tarde debido a incrementos en el retardo de conmutación del transistor de salida y tiempos de ascenso y caída de la señal. Por tanto, un diseñador cuidadoso intenta equilibrar las cargas en múltiples relojes, examinando tanto la carga de DC (fan out) como la carga de AC (capacitancia de entrada y alambrado).

Otra situación preocupante puede presentarse cuando las señales en un PCB o en un ASIC son dirigidas de forma automática por el software de CAD. La figura 8-87 ilustra un PCB o ASIC con muchos flip-flops y elementos de escala más grande, todos temporizados con una señal común CLOCK. El software CAD ha representado CLOCK en una trayectoria de serpentina que tuerce su camino pasando por todos los dispositivos temporizados. Otras señales son dirigidas punto a punto entre una salida y un número pequeño de entradas, de modo que sus trayectorias son más cortas. Para empeorar el asunto, en un ASIC algunos tipos de "alambrado" pueden ser más lentos que otros (polisilicio contra metal en la tecnología CMOS). Como resultado, una frontera de CLOCK puede en verdad llegar a FF2 un instante bastante más tarde que el cambio de los datos que produce en Q1.

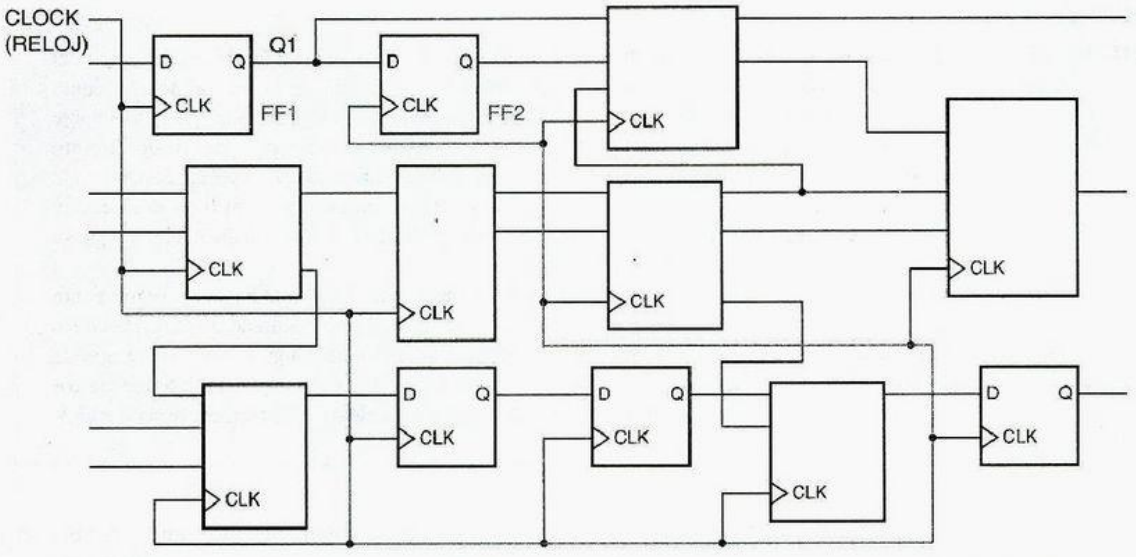
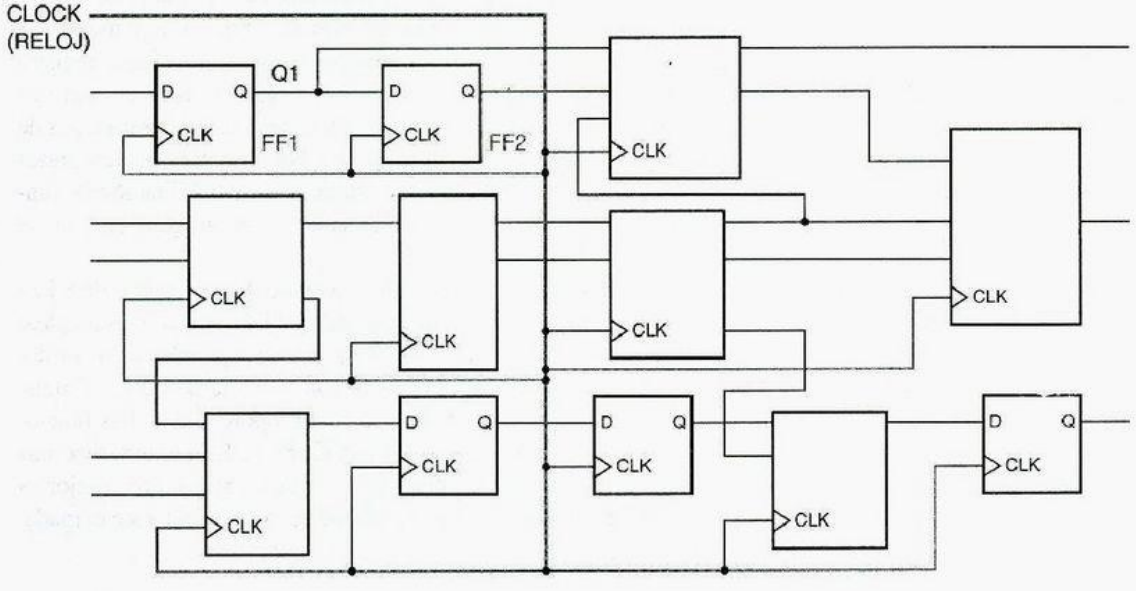


Figura 8-87 Una trayectoria de señal del reloj que conduce a sesgo excesivo en un ASIC o PCB complejo.

Una manera de minimizar esta clase de problema es hacer que CLOCK se distribuya en una estructura tipo árbol, para esto utilizamos el alambre que permita obtener la mayor rapidez en la transmisión de la información, como se ilustra en la figura 8-88. Por lo regular, un "árbol de reloj" de esta naturaleza debe ser trazado a mano o aplicando una herramienta especializada de CAD. Aun entonces, en un diseño completo puede no ser posible garantizar que los flancos de reloj lleguen a todos lados antes del primer cambio de datos. Un programa de análisis de temporización de CAD se utiliza habitualmente para detectar estos

Figura 8-88 Enrutamiento de señal de reloj para minimizar el sesgo.



CÓMO NO TENER SESGOS

Las cargas y longitudes de alambres no balanceadas son las fuentes más obvias del sesgo del reloj, pero existen muchas otras fuentes más sutiles. Por ejemplo, *crosstalk*, el acoplamiento de energía de una línea en otra, puede ocasionar un sesgo del reloj. El crosstalk es inevitable cuando se empacan cables paralelos estrechamente juntos sobre una tarjeta de circuito impreso o en un chip, y la energía es radiada durante las transiciones de señal. Dependiendo de si una señal adyacente se encuentra cambiando en la misma dirección o en la dirección opuesta como un reloj, la transición del reloj puede ser acelerada o retardada, haciendo que su transición parezca ocurrir más pronto o más tarde.

En un diseño grande PCB o ASIC, habitualmente no es factible hacer un seguimiento hacia atrás de todas las fuentes posibles de sesgo de reloj. Como resultado, la mayoría de los fabricantes de ASIC solicitan a sus diseñadores proporcionar margen extra de tiempo de establecimiento y retención, equivalente a muchos retardos de compuerta, además de los resultados conocidos de temporización de simulación para alojar tales factores desconocidos.

problemas, los cuales generalmente pueden remediarse solamente insertando un retardo extra (por ejemplo, pares de inversores) en las trayectorias de datos demasiado rápidas.

Aunque la metodología del diseño sincrónico simplifica el funcionamiento conceptual de sistemas grandes, observamos que el sesgo de reloj puede ser un problema principal cuando se utilizan flip-flops disparados por flanco como los elementos de almacenamiento. Para controlar este problema, muchos sistemas de alto rendimiento y chips VLSI hacen uso de un *diseño de latch de dos fases*, que se discute en las Referencias. Tales diseños efectivamente dividen cada flip-flop D disparado por flanco en dos latchs componentes, controlándolos con dos fases de reloj que no se traslapan. El no traslape entre las fases acomoda el sesgo de reloj.

diseño de latchs de dos fases

8.8.2 Disparo del reloj

La mayoría de las partes MSI secuenciales que introducimos en este capítulo tienen entradas habilitadoras de función sincrónicas. Es decir, sus entradas de habilitación se muestrean en el flanco de reloj, junto con los datos. El primer ejemplo que mostramos fue el registro 74x377 con entrada habilitadora de carga sincrónica; otras partes incluyen el contador 74x163 y el registro de corrimiento 74x194 con entradas sincrónicas de habilitación de carga, habilitación de contador y habilitación de corrimiento. No obstante, muchas partes MSI, macros FPGA y celdas ASIC no tienen entradas sincrónicas de habilitación de función; por ejemplo, el registro de 8 bits 74x374 tiene tres salidas pero ninguna entrada de habilitación de carga.

Así, ¿qué puede hacer un diseñador si una palabra requiere de un registro de 8 bits con una entrada de habilitación de carga y salidas de tres estados? Una solución es emplear un 74x377 para obtener la habilitación de carga, y seguirlo con un separador de tres estados 74x241. Sin embargo, esto incrementa tanto el costo como el retardo. Otro enfoque es utilizar una parte más grande y costosa, el 74x823, la cual proporciona ambas funciones requeridas así como también una entrada asincrónica CLR_L. Una alternativa más riesgosa que es empleada en ocasiones por los diseñadores que no saben algo mejor es utilizar un '374, pero suprimir su entrada de reloj cuando no se supone que esté cargada. Esto se denomina *disparo de reloj*.

disparo de reloj

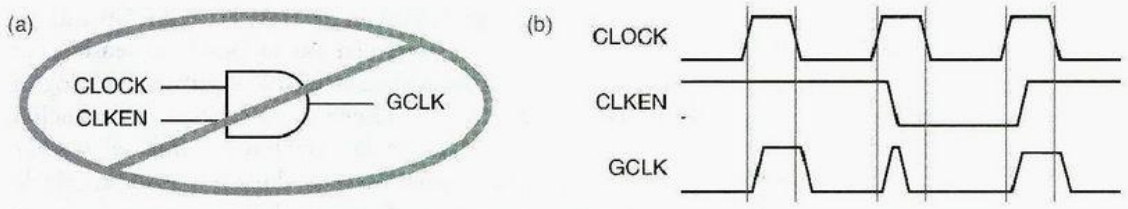


Figura 8-89 Cómo no disparar el reloj: (a) circuito simple; (b) diagrama de temporización.

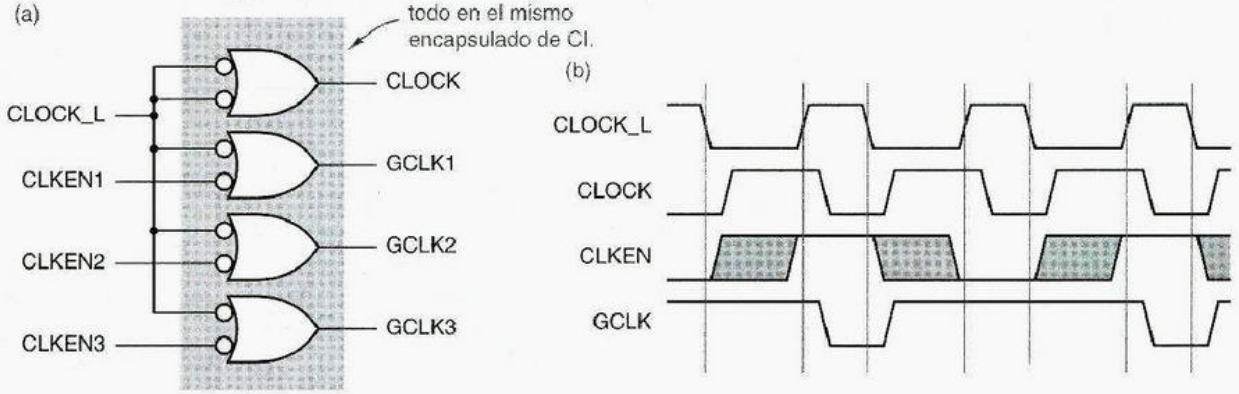
La figura 8-89 ilustra un enfoque obvio pero equivocado para disparo de reloj. Una señal CLKEN es asertiva para habilitar el reloj y simplemente es operada con AND con el reloj para producir la señal GCLK de reloj controlada por compuerta. Este enfoque tiene dos problemas:

1. Si CLKEN es una señal de máquina de estado u otra señal producida por un registro temporizado por CLOCK, entonces CLKEN cambia algún tiempo *después* de que CLOCK ya se ha hecho ALTO. Como se muestra en (b), esto produce interferencias en GCLK y temporización falsa de los registros controlados por GCLK.
2. Incluso si CLKEN es producido de una u otra manera con antelación de la frontera ascendente del CLOCK (por ejemplo, utilizando un registro temporizado con la frontera *de caída* de CLOCK, un asunto especialmente desagradable), el retardo de la compuerta AND proporciona a GCLK un sesgo de reloj excesivo, lo que ocasiona más problemas en total.

Un método para disparar el reloj que genera solamente un mínimo sesgo de reloj se ilustra en la figura 8-90. Aquí, se generan tanto un reloj no controlado por compuerta como relojes controlados por compuerta, a partir de la misma señal activa en estado bajo de reloj maestro. Las compuertas en el mismo encapsulado de CI se utilizan para minimizar las posibles diferencias en sus retardos. La señal CLKEN puede cambiar arbitrariamente cada vez que CLOCK_L esté en nivel BAJO, lo cual es cuando CLOCK se encuentra en nivel ALTO. Eso está muy bien; una señal CLKEN se produce habitualmente por una máquina de estado cuyas salidas cambian justo después de que CLOCK se va al nivel ALTO.

El enfoque de la figura 8-90 es aceptable en una aplicación particular solamente si el sesgo de reloj que crea es aceptable. Además, note que CLKEN debe estar estable

Figura 8-90 Una manera aceptable de disparar el reloj: (a) circuito; (b) diagrama de temporización.



durante todo el tiempo que CLOCK_L se encuentra en nivel ALTO (CLOCK está en nivel BAJO). De este modo, los márgenes de tiempo en este enfoque son sensibles al ciclo de trabajo del reloj, en especial si CLKEN sufre de un retardo significativo de lógica combinacional (t_{comb}) del flanco de reloj de disparo. Una entrada de habilitación de función verdaderamente sincrónica, tal como la entrada de habilitación de carga del 74x377 en la figura 8-13, puede cambiarse en casi cualquier momento durante todo el periodo de reloj, hasta un tiempo de establecimiento antes del flanco de disparo.

8.8.3 Entradas asincrónicas

Aun cuando es teóricamente posible construir un sistema de computadora que sea completamente sincrónico, no se podría hacer mucho con él, a menos que usted pudiera sincronizar sus golpes de teclas con un reloj de 500 MHz. Los sistemas digitales de todos tipos inevitablemente deben tratar con *señales de entrada asincrónicas* que no se encuentran sincronizadas con el reloj de sistema.

señal de entrada asíncrona

Las entradas asincrónicas son con frecuencia solicitudes de servicio (por ejemplo, las interrupciones en una computadora) o "banderas" de estado (por ejemplo, un recurso que ya se encuentra disponible). Tales entradas normalmente cambian lentamente en comparación con la frecuencia de reloj del sistema, y no necesitan ser reconocidas en un tic de reloj en particular. Si una transición se pierde en un tic de reloj, siempre puede ser detectada en el siguiente. Las velocidades de transición de señales asincrónicas pueden abarcar desde menos de una por segundo (los golpes de tecla de una mecanógrafa lenta) hasta 100 MHz o más (solicitudes de acceso para memoria compartida de sistema multiprocesador de 500 MHz).

sincronizador

Ignorando el problema de la metaestabilidad, es fácil construir un *sincronizador*, un circuito que muestrea una entrada asincrónica y produce una salida que satisface los tiempos de establecimiento y retención requeridos en un sistema sincrónico. Como se muestra en la figura 8-91, un flip-flop D muestrea la señal asincrónica para cada tic del reloj del sistema y produce una salida sincrónica que es válida durante el siguiente periodo de reloj.

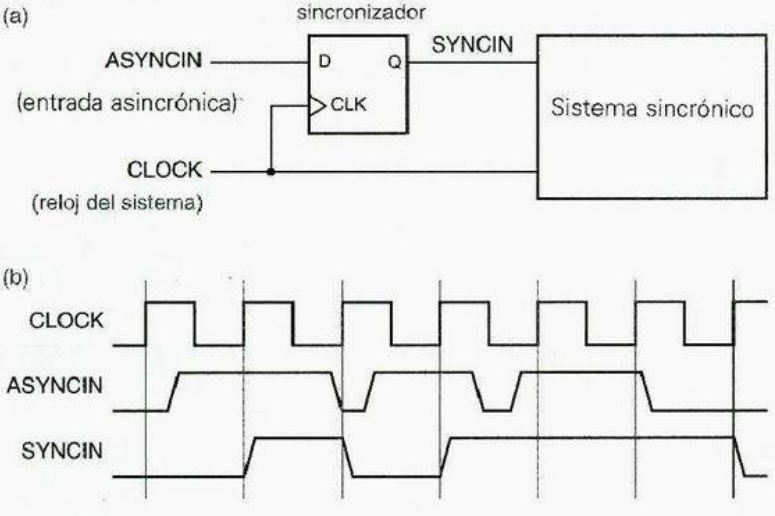


Figura 8-91
Un solo sincronizador simple: (a) diagrama lógico; (b) temporización.

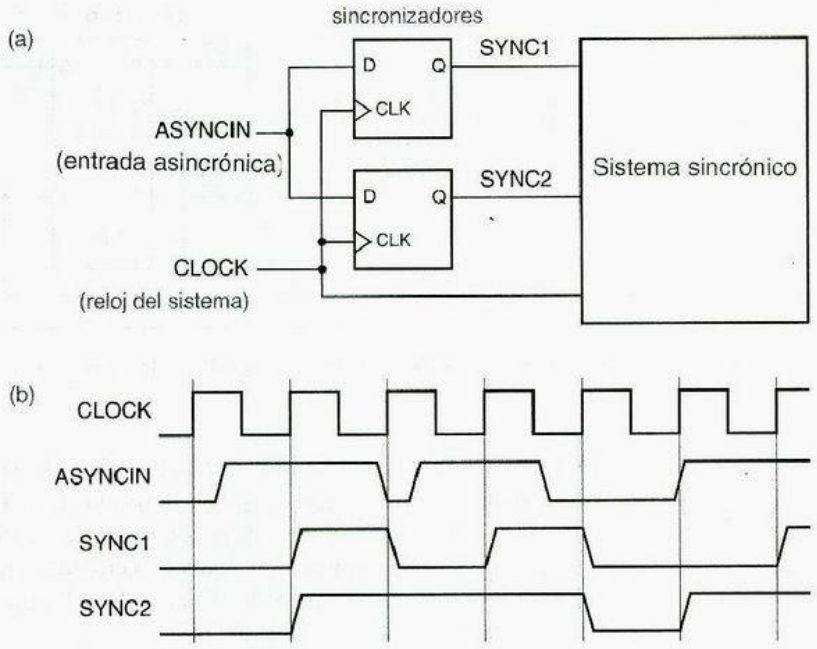
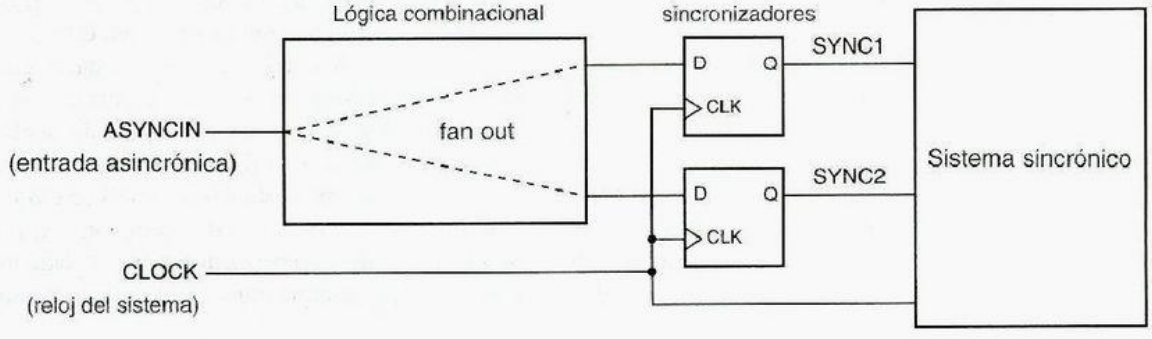


Figura 8-92
Dos sincronizadores para la misma entrada asincrónica: (a) diagrama lógico; (b) temporización posible.

Es esencial para las entradas asincrónicas estar sincronizadas solamente en *un lugar* en un sistema; la figura 8-92 muestra lo que puede ocurrir en caso contrario. Debido a los retardos físicos en el circuito; los dos flip-flops no verán las señales de entrada y reloj precisamente al mismo tiempo. Por lo tanto, cuando ocurren las transiciones de entrada asincrónicas cerca del flanco de reloj, hay una pequeña ventana de tiempo durante la cual un flip-flop puede muestrear la entrada como 1 y la otra puede muestrearla como 0. Este resultado inconsistente puede provocar un funcionamiento inapropiado del sistema, en cuanto que una parte del sistema responde como si la entrada fuera 1, y otra parte responde como si fuera 0.

La lógica combinacional puede ocultar el hecho de que existen dos sincronizadores, como se muestra en la figura 8-93. Puesto que trayectorias diferentes a través de la lógica combinacional inevitablemente tendrán diferentes retardos, la probabilidad

Figura 8-93 Una entrada asincrónica controlando dos sincronizadores a través de lógica combinacional.



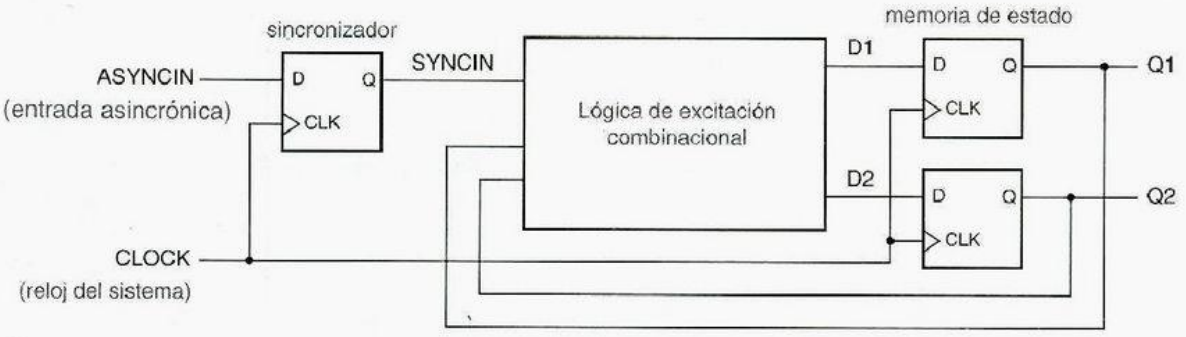


Figura 8-94 Una entrada de máquina de estado asincrónica conectada a través de un sincrónico simple.

de un resultado inconsistente es todavía más grande. Esta situación es en especial común cuando las señales asincrónicas se utilizan como entradas a máquinas de estado, puesto que la lógica de excitación para dos o más variables de estado puede depender de la entrada asincrónica. La manera apropiada de emplear una señal asincrónica como una entrada de máquina de estado se exhibe en la figura 8-94. Todo lo de la lógica de excitación ve la misma señal de entrada sincronizada, SYNCIN.

¿A QUIÉN LE IMPORTA? Como usted probablemente sabrá, incluso los sincronizadores en las figura 8-91 y 8-94 a veces fallan. La razón es que los tiempos de establecimiento y retención del flip-flop (dispositivo biestable) de sincronización son violados en ocasiones, debido a que la entrada asincrónica puede cambiar en cualquier momento. "Bueno, ¿y a quién le importa?", usted podría decir. "Si la entrada D cambia cerca de la frontera del reloj, entonces el flip-flop (dispositivo biestable) verá el cambio en este momento o lo perderá y lo obtendrá la siguiente vez, ¡de cualquier manera es suficientemente bueno para mí!" El problema es que existe una tercera posibilidad, discutida en la siguiente sección.

8.9 Falla del sincronizador y metaestabilidad

Mostramos en la sección 7.1 que cuando los tiempos de establecimiento y de retención de un flip-flop no se satisfacen, el flip-flop puede entrar a un tercer estado *metaestable* a medio camino entre 0 y 1. Peor aún, el tiempo que puede permanecer en este estado antes de caer de regreso a un estado legítimo de 0 o 1 teóricamente no está acotado. Cuando a otras compuertas y flip-flops se les presenta una señal de entrada metaestable, algunas pueden interpretarlo como un 0 y otras como un 1, creando la clase de comportamiento inconsistente que hemos mostrado en la figura 8-92. O las mismas otras compuertas y flip-flops pueden producir salidas metaestables (después de todo, se encuentran ahora funcionando en la parte *lineal* de su intervalo de operación). Afortunadamente, la probabilidad de que la salida de un flip-flop permanezca en el estado metaestable decrece exponencialmente con el tiempo, aunque nunca se reduce del todo a cero.

8.9.1 Falla de sincronizador

falla del sincronizador

Se dice que ocurre una *falla del sincronizador* si un sistema utiliza una salida de sincronizador mientras la salida todavía se encuentra en el estado metaestable. La manera de evitar la falla del sincronizador es asegurarse de que el sistema espere "lo suficiente" antes de emplear una salida de sincronizador, "lo suficiente" de tal modo que el tiempo medio entre fallas de sincronizador sea de varios órdenes de magnitud mayor que la duración esperada del empleo del diseñador.

La metaestabilidad es más que un problema académico. Más de unos cuantos diseñadores con experiencia en sistemas digitales de alta velocidad han construido (y liberado para la producción) circuitos que sufren de fallas intermitentes de sincronización. De hecho, se dice que las versiones iniciales de varios circuitos integrados comerciales han tenido problemas de metaestabilidad. Por ejemplo, el controlador de temporización de sistema AMD 9513, el controlador de interrupción AMD 9519, la interfaz de E/S serie Zilog Z-80, el chip microcomputador Intel 8048 y el microprocesador RISC AMD 29000. Esto hace pensar, ¿acaso los diseñadores de estas partes conservan aún sus empleos?

Existen dos maneras de librar a un flip-flop del estado metaestable:

1. Forzar al flip-flop a un estado lógico válido que utiliza señales de entrada que satisfacen las especificaciones de anchura de pulso mínima, tiempo de establecimiento, y así sucesivamente.
2. Esperar "lo suficiente", de modo que el flip-flop salga de la metaestabilidad por sí solo.

Los diseñadores sin experiencia con frecuencia intentan evitar la metaestabilidad en otras formas, y por lo regular no tienen éxito. La figura 8-95 muestra el intento de un diseñador que piensa que ya que la metaestabilidad es un problema "analógico", debe tener una solución "analógica". Después de todo, los capacitores y entrada de disparador Schmitt pueden normalmente ser utilizadas para limpiar señales ruidosas. Sin embargo, más que eliminar la metaestabilidad, este circuito la incrementa: con los componentes "correctos", el circuito oscilará eternamente, una vez que es excitado al negar S_L y R_L simultáneamente. (*Confesión*: ¡fue el autor quien intentó esto hace ya 20 años!). Los ejercicios 8.97 y 8.94 proporcionan ejemplos de intentos meritorios pero fallidos para eliminar la metaestabilidad. Estos ejemplos deberían darle a usted la sensación de que los problemas de sincronizador pueden ser muy sutiles, de manera que usted debe ser cuidadoso. La única manera de hacer sincronizadores confiables es esperar lo suficiente para resolver salidas metaestables. Contestaremos la pregunta ¿cuánto tiempo es "lo suficiente"? más adelante en esta sección.

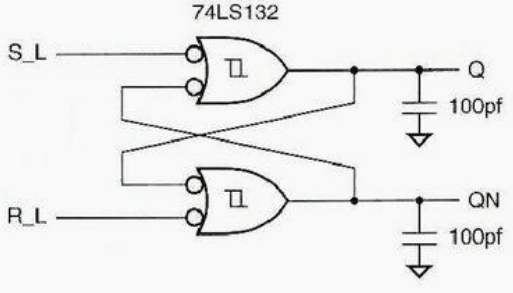


Figura 8-95
Un intento fallido para construir un flip-flop (dispositivo biestable) $\overline{S}$-$\overline{R}$ a prueba de metaestabilidad.

8.9.2 Tiempo de resolución de la metaestabilidad

Si los tiempos de establecimiento y retención de un flip-flop D se satisfacen, la salida del flip-flop se establece a un nuevo valor dentro del tiempo t_{pd} después del flanco de reloj. Si son violados, la salida del flip-flop puede ser metaestable por una extensión arbitraria de tiempo. En un diseño de sistema particular, utilizamos el parámetro t_r, denominado *tiempo de resolución de metaestabilidad*, para indicar el tiempo máximo en que la salida puede permanecer metaestable sin ocasionar una falla del sincronizador (y del sistema).

t_r
tiempo de resolución de metaestabilidad

Por ejemplo, considere la máquina de estado de la figura 8-94. El tiempo de resolución de metaestabilidad es

$$t_r = t_{clk} - t_{comb} - t_{establecimiento}$$

donde t_{clk} es el periodo del reloj, t_{comb} es el retardo de propagación de la lógica de excitación combinacional, y $t_{establecimiento}$ es el tiempo de establecimiento de los flip-flops (dispositivos biestables) empleados en la memoria de estado.

t_{clk}
t_{comb}
$t_{establecimiento}$

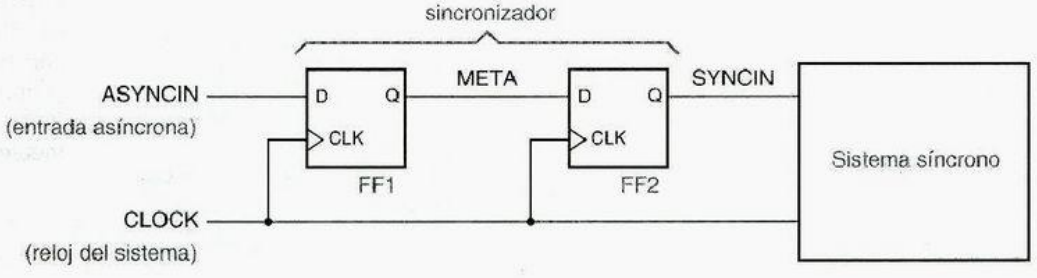
8.9.3 Diseño de sincronizador confiable

El sincronizador más confiable es aquel que permite la máxima cantidad de tiempo para la resolución de la metaestabilidad. Sin embargo, en el diseño de un sistema digital, rara vez tenemos el lujo de *demorar* el reloj para hacer que el sistema trabaje de manera más confiable. En su lugar, habitualmente solicitamos *acelerar* el reloj para obtener un mayor rendimiento del sistema. Como resultado, a menudo necesitamos sincronizadores que funcionen confiablemente con periodos muy breves de reloj. Presentaremos varios de estos diseños, y mostraremos cómo predecir su confiabilidad.

Mostramos con anterioridad que una máquina de estado con una entrada asincrónica, construida como se ilustra en la figura 8-94, tiene $t_r = t_{clk} - t_{comb} - t_{establecimiento}$. Con el fin de maximizar t_r para un periodo de reloj dado, deberíamos minimizar t_{comb} y $t_{establecimiento}$. El valor de $t_{establecimiento}$ depende del tipo de flip-flop que se utiliza en la memoria de estado; en general, los flip-flops más rápidos tienen tiempos de establecimiento más cortos. El valor mínimo de t_{comb} es cero y se consigue mediante el diseño del sincronizador de la figura 8-96, cuya operación explicamos a continuación.

Las entradas del flip-flop FF1 son asincrónicas con el reloj y pueden violar los tiempos de establecimiento y retención. Cuando esto ocurre, la salida META puede llegar a ser metaestable y permanecer en ese estado durante un tiempo. Sin embargo, suponemos

Figura 8-96 Diseño recomendado de sincronizador.



que la máxima duración de la metaestabilidad después del flanco de reloj es t_r. (Mostraremos cómo calcular la probabilidad de que nuestra suposición sea correcta en la subsección siguiente). Mientras el periodo de reloj es mayor que t_r más el tiempo de establecimiento de FF2, SYNCIN se convierte en una copia sincronizada de la entrada asincrónica en el siguiente tic de reloj sin llegar a ser metaestable ella misma. La señal SYNCIN se distribuye como se requiere al resto del sistema.

8.9.4 Análisis de la temporización metaestable

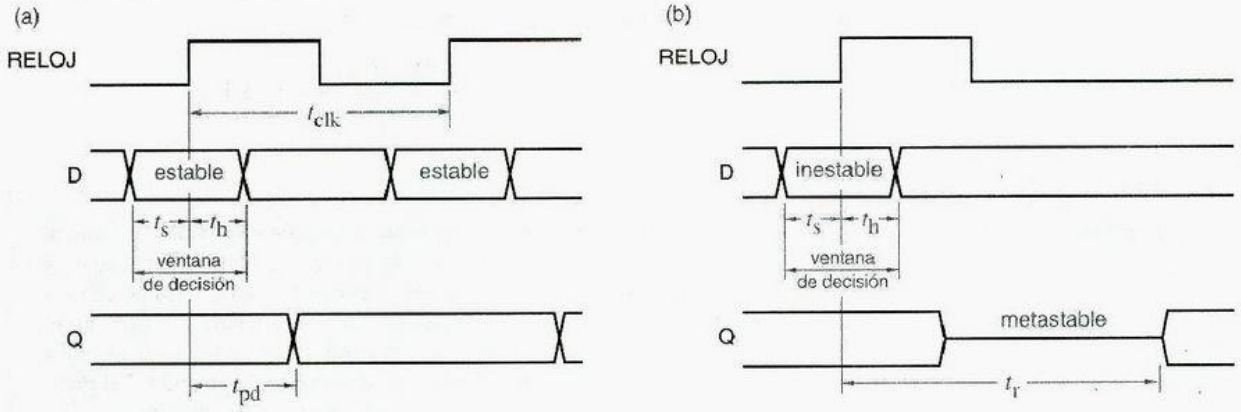
La figura 8-97 muestra los parámetros de temporización del flip-flop que son relevantes para nuestro análisis de temporización de metaestabilidad. Los tiempos publicados de establecimiento y retención de un flip-flop con respecto a su flanco de reloj se indican por t_s y t_h y delimitan un intervalo llamado la *ventana de decisión*, cuando el flip-flop muestrea su entrada y decide cambiar su salida si es necesario. Mientras la entrada D cambia fuera de la ventana de decisión, como en (a), el fabricante garantiza que la salida cambiará y establecerá un estado lógico válido antes del tiempo t_{pd}. SI D cambia dentro de la ventana de decisión, como en (b), puede presentarse metaestabilidad y persistir hasta el tiempo t_r.

ventana de decisión

La investigación teórica sugiere, confirmándose con la investigación experimental, que cuando las entradas asincrónicas cambian durante la ventana de decisión, la duración de las salidas metaestables está gobernada por una fórmula exponencial:

$$\text{MTBF}(t_r) = \frac{\exp(t_r/\tau)}{T_o \cdot f \cdot a}$$

Figura 8-97 Parámetros de temporización para análisis de metaestabilidad: (a) funcionamiento normal de flip-flop, (b) comportamiento metaestable.



DETALLES, DETALLES En nuestro análisis del sincronizador de la figura 8-96 no permitimos metaestabilidad, incluso brevemente, en la salida de FF2, porque supusimos que el sistema había sido diseñado con cero márgenes de temporización. Si el sistema puede de hecho tolerar algún incremento en el retardo de propagación de FF2, el MTBF será algo mejor del que se predijo.

Aquí MTBF(t_r) es el tiempo medio entre fallas de sincronizador, donde una falla ocurre si persiste la metaestabilidad más allá del tiempo t_r después de una frontera de reloj, donde $t_r \geq t_{pd}$. Este MTBF depende de f, la frecuencia del reloj del flip-flop; a, el número de cambios de entrada asincrónica por segundo aplicados al flip-flop; y T_0 y τ, constantes que dependen de las características eléctricas del flip-flop. Para un 74LS74 típico, $T_0 \approx 0.4$ s y $\tau \approx 1.5$ ns.

f
a
T_0
τ

Ahora supongamos que construimos un sistema microprocesador con un reloj de 10 MHz y utilizamos el circuito de la figura 8-96 para sincronizar una entrada asincrónica. Si ASYNCIN cambia durante la ventana de decisión de FF1, la salida META puede llegar a ser metaestable hasta el tiempo t_r. META es todavía metaestable al principio de la ventana de decisión para FF2, entonces el sincronizador fallará, porque FF2 puede tener una salida metaestable; el funcionamiento del sistema será impredecible en este caso.

Supongamos que los flip-flops D en la figura 8-96 son 74LS74s. El tiempo de establecimiento t_s de un 74LS74 es de 20 ns, y el periodo de reloj en nuestro sistema microprocesador de ejemplo es 100 ns, de modo que t_r para la falla del sincronizador es de 80 ns. Si la entrada asincrónica cambia 100,000 veces por segundo, entonces el MTBF del sincronizador es

$$\text{MTBF}(80\text{ ns}) = \frac{\exp(80/1.5)}{0.4 \cdot 10^7 \cdot 10^5} = 3.6 \cdot 10^{11}\text{s}$$

Esto no es malo, aproximadamente ¡100 siglos entre fallas! Naturalmente, si somos lo suficientemente afortunados para vender 10,000 copias de nuestro sistema, uno de ellos fallará de esta manera cada año. Pero, no importa, consideremos un problema más serio.

Supongamos que actualizamos nuestro sistema para utilizar un chip microprocesador más rápido con una velocidad de reloj de 16 MHz. Podemos tener que reemplazar algunos componentes en nuestro sistema para funcionar a mayor velocidad, pero los 74LS74 son todavía perfectamente capaces de operar a 16 MHz. ¿O no? Con un periodo de reloj de 62.5 ns, el nuevo MTBF del sincronizador es

$$\text{MTBF}(42.5\text{ ns}) = \frac{\exp(42.5/1.5)}{0.4 \cdot 1.6 \cdot 10^7 \cdot 10^5} = 3.1\text{ s!}$$

COMPRENDIENDO *A Y F*

Aunque una salida de flip-flop puede hacerse metaestable *solamente* si D cambia durante la ventana de decisión, la fórmula MTBF no especifica de manera explícita cuántos de esos cambios de entrada ocurren. En su lugar, especifica el número total de cambios de entrada asincrónica por segundo, a, y supone que los cambios de entrada asincrónica se encuentran uniformemente distribuidos sobre el periodo del reloj. Por lo tanto, la fracción de cambios de entrada que ocurren en realidad durante la ventana de decisión está "integrada" al parámetro f de frecuencia del reloj; a medida que f se incrementa, la fracción aumenta.

Si el diseño del sistema permite agrupar los cambios de entrada en la ventana de decisión, en vez de distribuirlos en forma uniforme (como sucede cuando se sincroniza una entrada lenta con una diferencia de fase fija pero desconocida del reloj del sistema), entonces una útil y práctica regla es emplear una frecuencia igual al recíproco de la ventana de decisión (basada en los tiempos de establecimiento y retención publicados), por un margen de seguridad de 10.

¡El único mérito de este sincronizador a 16 MHz es que es tan fatal que probablemente descubramos el problema en el laboratorio de ingeniería antes de que se embarque el producto! Gracias a Dios que el MTBF no era de un año.

8.9.5 Mejores sincronizadores

Dado el pobre rendimiento del 74LS74 como un sincronizador a velocidades moderadas de reloj, tenemos un par de alternativas para construir sincronizadores más confiables. La solución más simple, la cual funciona con la mayoría de los requerimientos de diseño, es simplemente utilizar un flip-flop de una tecnología más rápida. En la actualidad se encuentran disponibles muchas tecnologías más rápidas para flip-flops, ya sean discretos o incorporados en los PLD, en los FPGA o en los ASIC.

Basados en los datos que se analizan en las Referencias, en la tabla 8-35 enumeramos los parámetros de metaestabilidad para varios dispositivos y familias lógicas comunes. Estos números son mucho más dependientes del diseño de circuito y del proceso de CI. De este modo, a diferencia de los parámetros de temporización y niveles de señal lógicos garantizados, los números de metaestabilidad publicados pueden variar dramáticamente entre diferentes fabricantes de la misma parte y deben utilizarse de manera conservadora. Por ejemplo, un 74F74 de fabricante puede dar un rendimiento de metaestabilidad aceptable en un diseño mientras que otro no.

Tabla 8-35 Parámetros de metaestabilidad para algunos dispositivos comunes.

Referencia	*Dispositivo*	τ *(ns)*	T_o *(s)*	t_r *(ns)*
Chaney (1983)	74LS74	1.50	$4.0 \cdot 10^{-1}$	77.71
Chaney (1983)	74S74	1.70	$1.0 \cdot 10^{-6}$	66.14
Chaney (1983)	74S174	1.20	$5.0 \cdot 10^{-6}$	48.62
Chaney (1983)	74S374	0.91	$4.0 \cdot 10^{-4}$	40.86
Chaney (1983)	74F74	0.40	$2.0 \cdot 10^{-4}$	17.68
TI (1997)	74LSxx	1.35	$4.8 \cdot 10^{-3}$	63.97
TI (1997)	74Sxx	2.80	$1.3 \cdot 10^{-9}$	90.33
TI (1997)	74ALSxx	1.00	$8.7 \cdot 10^{-6}$	41.07
TI (1997)	74ASxx	0.25	$1.4 \cdot 10^{3}$	14.99
TI (1997)	74Fxx	0.11	$1.9 \cdot 10^{8}$	7.90
TI (1997)	74HCxx	1.82	$1.5 \cdot 10^{-6}$	71.55
Cypress (1997)	PALC16R8-25	0.52	$9.5 \cdot 10^{-12}$	14.22*
Cypress (1997)	PALC22V10B-20	0.26	$5.6 \cdot 10^{-11}$	7.57*
Cypress (1997)	PALCE22V10-7	0.19	$1.3 \cdot 10^{-13}$	4.38*
Xilinx (1997)	Serie 7300 CPLD	0.29	$1.0 \cdot 10^{-15}$	5.27*
Xilinx (1997)	Serie 9500 CPLD	0.17	$9.6 \cdot 10^{-18}$	2.30*

Advierta que autores y fabricantes diferentes pueden especificar parámetros de metaestabilidad de forma diferente. Por ejemplo, el autor Chaney y el fabricante Texas Instruments miden el tiempo de resolución de la metaestabilidad t_r desde la frontera de reloj de disparo, como en nuestra subsección anterior. Por otra parte, los fabricantes Cypress y Xilinx definen t_r como el retardo *adicional* más allá de un tiempo de retardo normal de reloj a salida t_{pd}.

La última columna en la tabla proporciona un estimado elegido arbitrariamente para cada dispositivo. Es el tiempo de resolución de metaestabilidad t_r requerido para obtener un MTBF de 1000 años cuando se hace funcionar un sincronizador con una frecuencia de reloj de 25 MHz y con 100,000 cambios de entrada asincrónica por segundo. Para los dispositivos Cypress y Xilinx, sus valores de parámetro conducen a un valor de t_r, marcado con un asterisco, consistente con sus propias definiciones como se presentaron con anterioridad.

Como puede verse, el 74LS74 es uno de los peores dispositivos en la tabla. Si reemplazamos FF1 en el sistema microprocesador de 16 MHz de la subsección anterior con un 74ALS74, obtenemos

$$\text{MTBF}(42.5\text{ ns}) = \frac{\exp(42.5/1.00)}{8.7 \cdot 10^{-6} \cdot 1.6 \cdot 10^{7} \cdot 10^{5}} = 2.06 \cdot 10^{11}\text{s}$$

Si usted se encuentra satisfecho con un MTBF de sincronizador de aproximadamente 65 siglos por sistema embarcado, puede detenerse en este punto. Sin embargo, si FF2 también es reemplazado por un 74ALS74, el MTBF mejora, puesto que el ALS74 tiene un tiempo de establecimiento más corto que el LS74, solamente de 10 ns. Con el ALS74, el MTBF es aproximadamente 100,000 veces mejor:

$$\text{MTBF}(52.5\text{ ns}) = \frac{\exp(52.5/1.00)}{8.7 \cdot 10^{-6} \cdot 1.6 \cdot 10^{7} \cdot 10^{5}} = 4.54 \cdot 10^{15}\text{s}$$

Incluso si embarcamos un millón de sistemas conteniendo este circuito, nosotros (o nuestros herederos) veremos una falla del sincronizador solamente una vez en 144 años. ¡Esto es seguridad en el trabajo!

En realidad los márgenes anteriores no son tan grandes como parecen. (¿Qué tan largos parecen 144 años *para usted*?). La mayoría de los números dados en la tabla 8-35 son *promedios* y rara vez son especificados, y mucho menos garantizados, por el fabricante del dispositivo. Además, los MTBF calculados son extremadamente sensibles al valor de τ, el cual a su vez depende de la temperatura, el voltaje y las fases de la luna. De manera que el funcionamiento de un flip-flop dado en un sistema real puede ser mucho peor (o mucho mejor) que el que se predice mediante nuestra tabla.

Por ejemplo, considere lo que ocurre si incrementamos el reloj de nuestro sistema en un 25%, a 20 MHz. Su inclinación natural sería pensar que la metaestabilidad se hará un 25% peor, o tal vez 250% peor, sólo para ser conservadores. Pero, si se trabajan los números, se encontrará que el MTBF, cuando se utiliza un dispositivo 'ALS74 (tanto para FF1 como para FF2), desciende de $4.54 \cdot 10^{15}$ a sólo $3.7 \cdot 10^{9}$ s. ¡Alrededor de un millón de veces peor! El nuevo MTBF de cerca de 429 años es adecuado para un sistema, pero si usted embarca un millón de ellos, uno fallaría cada hora. ¡Usted acaba de pasar de generaciones de seguridad en el trabajo a un fastidio colectivo!

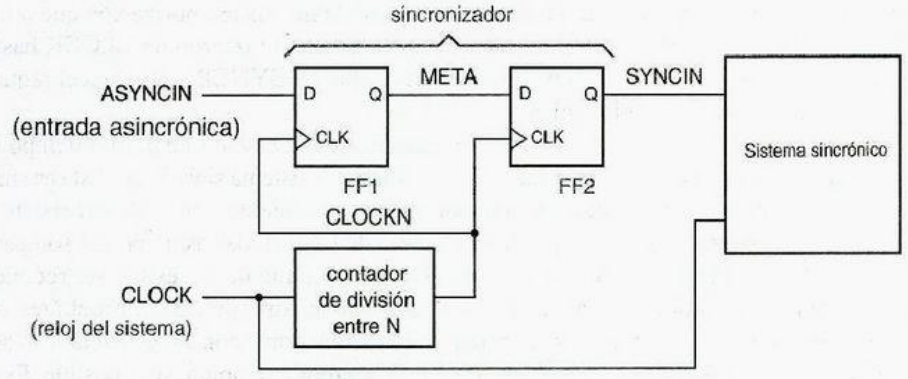


Figura 8-98
Sincronizador de ciclo múltiple.

8.9.6 Otros diseños de sincronizador

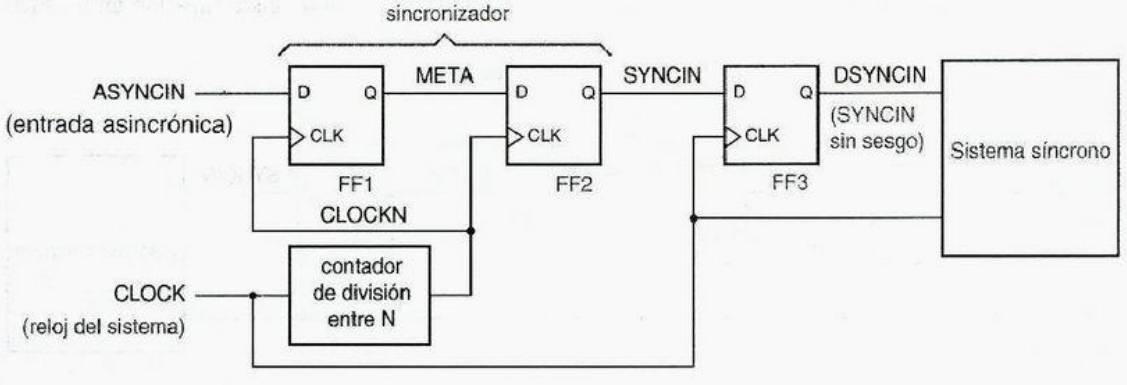
Prometimos describir un par de maneras para construir sincronizadores más confiables. La primera manera era utilizar flip-flops (dispositivos biestables) más rápidos, es decir, reducir el valor de τ en la ecuación MTBF. Habiendo dicho esto, la segunda manera es obvia: *incrementar* el valor de t_r en la ecuación MTBF.

Para un reloj de sistema dado, el mejor valor que podemos obtener para t_r (si utilizamos el circuito de la figura 8-96) es t_{clk}, si FF2 tiene un tiempo de establecimiento de 0. Sin embargo, podemos obtener valores de t_r en el orden de $n \cdot t_{clk}$ mediante el uso del circuito *sincronizador de ciclo múltiple* de la figura 8-98. Aquí dividimos el reloj del sistema entre n para obtener un reloj sincronizador más lento y mayor $t_r = (n \cdot t_{clk}) - t_{establecimiento}$. Por lo regular un valor de $n = 2$ o $n = 3$ proporciona una adecuada confiabilidad del sincronizador.

sincronizador de ciclo múltiple

En la figura, advierta que los flancos de CLOCKN retrasarán los flancos de CLOCK debido a que CLOCKN viene desde la salida Q de un flip-flop contador que está temporizado por CLOCK. Esto significa que SYNCIN, a su vez, será retrasada o sesgada en relación con otras señales en el sistema sincrónico que vienen directamente de flip-flops temporizados por CLOCK. Si SYNCIN pasa a través de lógica combinacional adicional en el sistema sincrónico antes de que alcance sus entradas de flip-flop, su tiempo de establecimiento puede ser inadecuado. Si éste es el caso, puede utilizarse la solución en la figura 8-99. Aquí, SYNCIN vuelve a temporizarse por CLOCK

Figura 8-99 Sincronizador de ciclo múltiple con eliminación de sesgo.



empleando FF3 para producir DSYNCIN, que tendrá la misma temporización que otras salidas de flip-flop en el sistema sincrónico. Por supuesto, el retardo de CLOCK hasta CLOCKN debe ser todavía suficientemente corto para que SYNCIN satisfaga el requerimiento de tiempo de establecimiento de FF3.

En un sincronizador de n ciclos, entre más grande sea el valor de n, más tiempo le toma a una entrada asincrónica cambiar para ser vista por el sistema sincrónico. Éste es simplemente el precio que tiene que pagarse por un funcionamiento confiable del sistema. En sistemas de microprocesador típicos, la mayoría de las entradas asincrónicas son para eventos externos-interrupciones, solicitudes DMA, etc., que no necesitan ser reconocidas muy rápidamente, en relación con los retardos del sincronizador. En el área de tiempo crítica del acceso a memoria principal, los diseñadores con experiencia utilizan el reloj del procesador para correr el subsistema de memoria también, si es posible. Esto elimina la necesidad de sincronizadores y proporciona el funcionamiento más rápido posible del sistema.

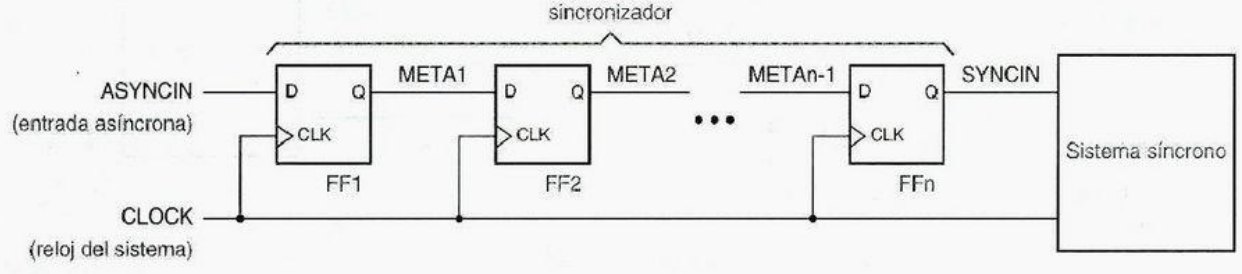
A frecuencias mayores, la viabilidad del diseño de sincronizador de ciclo múltiple mostrado en la figura 8-98 tiende a estar limitado por el sesgo de reloj. Por esta razón, más que utilizar un reloj de sincronizador de división entre n, algunos diseñadores emplean *sincronizadores en cascada*. Este enfoque de diseño simplemente hace uso de una cascada (registro de corrimiento) de n flip-flops, todos temporizados con el reloj del sistema de alta velocidad. Este enfoque se muestra en la figura 8-100.

sincronizadores en cascada

Con los sincronizadores en cascada, la idea es que la metaestabilidad será resuelta con alguna probabilidad mediante él primer flip-flop, y si falla éste, con una probabilidad igual para cada flip-flop sucesivo en la cascada. De modo que la probabilidad total de falla se encuentra en el orden de n ésima potencia de la probabilidad de falla de un sincronizador de un solo flip-flop trabajando a la frecuencia de reloj del sistema. Mientras que esto es parcialmente cierto, el MTBF de la cascada es más pobre que el de un sincronizador de ciclo múltiple con el mismo retardo ($n \cdot t_{clk}$). Con la cascada, el tiempo de establecimiento $t_{establecimiento}$ del flip-flop debe ser restado de t_r, el tiempo de resolución de metaestabilidad disponible, n veces, pero en un diseño de ciclo múltiple, se resta solamente una vez.

Los PLD que contienen flip-flops internos pueden emplearse en diseños de sincronizador, donde los dos flip-flops de la figura 8-96 están simplemente incluidos en el PLD. Esto es muy conveniente en la mayoría de las aplicaciones, porque elimina la necesidad de flip-flops discretos, externos. Sin embargo, un sincronizador basado en PLD habitualmente tiene un MTBF más pobre que un circuito discreto construido con una tecnología similar o idéntica. Esto ocurre debido a que cada flip-flop en un PLD

Figura 8-100 Sincronizador en cascada.



tiene un arreglo lógico combinacional en su entrada D que incrementa su tiempo de establecimiento y por ello reduce la cantidad de tiempo t_r disponible para la resolución de la metaestabilidad durante un periodo de reloj de sistema dado t_{clk}. Para maximizar t_r sin hacer uso de componentes especiales, FF2 en la figura 8-96 debería ser un flip-flop discreto con un tiempo de establecimiento corto.

8.9.7 Flip-flops metaestables endurecidos

A finales de la década de los ochenta, Texas Instruments y otros fabricantes desarrollaron flip-flops SSI y MSI que están específicamente diseñados para aplicaciones de sincronizador al nivel de tarjeta. Por ejemplo, el *74AS4374* era semejante al 74AS374, excepto que cada flip-flop individual era reemplazado con un par de flip-flops, como se muestra en la figura 8-101. Cada par de flip-flops podría utilizarse como un sincronizador del tipo ilustrado en la figura 8-96, de modo que ocho entradas asincrónicas podrían ser sincronizadas con un 74AS4374.

74AS4374

El diseño interno del 'AS4374 fue mejorado para reducir τ y T_0 en comparación con otros flip-flops 74AS, pero la mejora más grande en el 'AS4374 fue un $t_{establecimiento}$ enormemente reducido. Debido a que el sincronizador entero de la figura 8-96 se construye en un solo chip, no hay separadores de entrada o salida entre FF1 y FF2, y $t_{establecimiento}$ para FF2 es de solamente 0.5 ns. En comparación con un convencional flip-flop 74AS con un $t_{establecimiento}$ de 5 ns, y suponiendo que $\tau = 0.40$ ns, esto mejora el MTBF por un factor de exp(4.5/0.40), o de alrededor de 77,000.

En años recientes, el movimiento hacia tecnologías CMOS más rápidas y con mayor integración ha hecho obsoletas principalmente partes especializadas como el 'AS4374. Como se puede ver de los últimos renglones de la tabla 8-35, los PLD y CPLD rápidos están disponibles con valores de τ que rivalizan con los dispositivos discretos más rápidos mientras que ofrecen la conveniencia de la integración de sincronización con muchas otras funciones. Con todo, el enfoque empleado por 'AS4374 es emulado con mérito en diseños FPGA y ASIC. Esto es, donde usted tenga control sobre el esquema de un circuito sincronizador, para localizar FF1 y FF2 tan cercanos como sea posible entre sí, y conectarlos con los alambres más rápidos disponibles, a fin de maximizar el tiempo de establecimiento disponible para FF2.

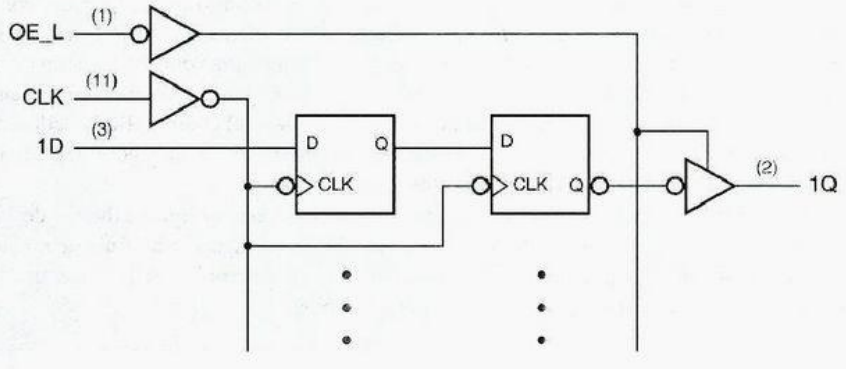


Figura 8-101
Diagrama lógico para el flip-flop D de doble rango octal 74AS4374.

8.9.8 Sincronización de transferencias de datos de alta velocidad

Un problema muy común en sistemas de computadoras es la sincronización de transferencias de datos externos con el reloj de sistema de la computadora. Un ejemplo simple es la interfaz entre la tarjeta de interfaz de red de una computadora personal y un enlace Ethernet de 100 Mbps. La tarjeta de interfaz puede ser conectada a un bus PCI, el cual tiene un reloj de 33.33 MHz. Aun cuando la velocidad Ethernet es un múltiplo aproximado de la velocidad del bus, la señal recibida en el enlace Ethernet es generada por otra computadora cuyo reloj transmisor no está sincronizado con el reloj receptor. Todavía la interfaz debe aún entregar datos de manera confiable al bus PCI.

La figura 8-102 muestra el problema. RDATA datos en serie NRZ son recibidos desde la Ethernet a 100 Mbps. El circuito digital de enganche o sincronización de fase (DPLL, digital phase-locked loop) recobra una señal de reloj de 100 MHz RCLK que está centrada en el flujo de datos de 100 Mbps y permite que los datos sean temporizados

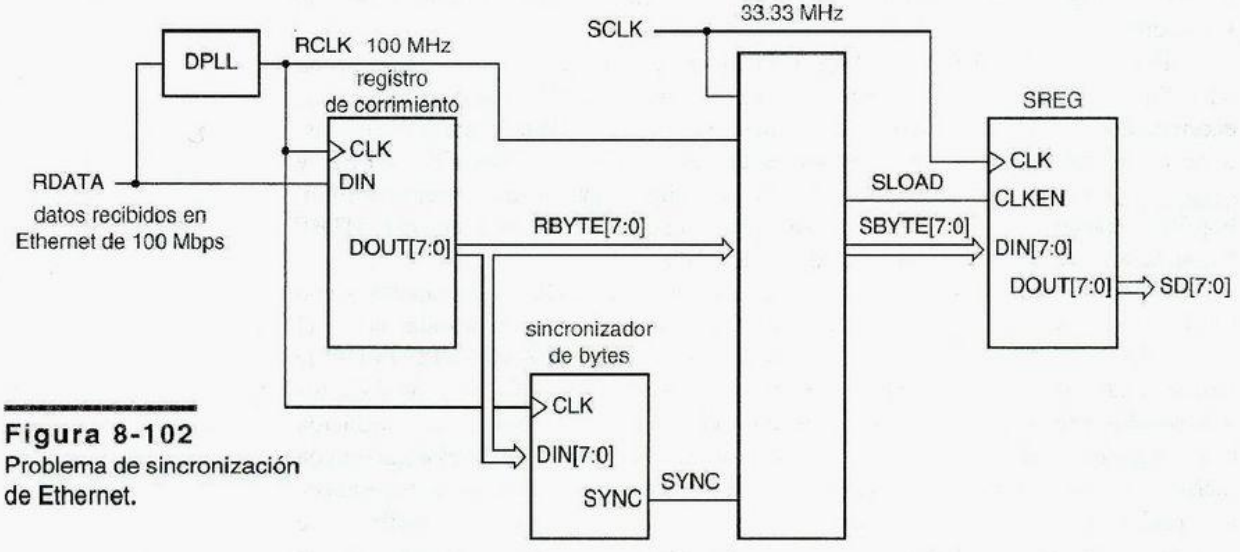


Figura 8-102
Problema de sincronización de Ethernet.

UN NIBBLE (MEDIO BYTE) A LA VEZ

La explicación de la recepción Ethernet de 100 Mbps anterior está sobresimplificada, pero es suficiente para discutir el problema de sincronización. En realidad, la velocidad de los datos recibidos es de 125 Mbps, donde cada 4 bits de datos del usuario son codificados como un símbolo de 5 bits utilizando un código denominado 4B5B. Mediante el uso de solamente 16 de 32 posibles palabras código de 5 bits, el código 4B5B garantiza que sin tener en cuenta el patrón de datos del usuario, el flujo de datos en el alambre tendrá un número suficiente de transiciones para permitir la recuperación del reloj. También, el código 4B5B incluye un código especial que es transmitido periódicamente para permitir que la sincronización de un nibble (4 bytes) y la de un byte se realice muy fácilmente.

Como resultado de la sincronización de un nibble, una interfaz típica Ethernet de 100 Mbps no ve un flujo de bits de 100 MHz desincronizado. En su lugar, ve un flujo de nibbles de 25 MHz desincronizado. De este modo, los detalles de un sincronizador Ethernet de 100 Mbps son diferentes, pero se aplican los mismos principios.

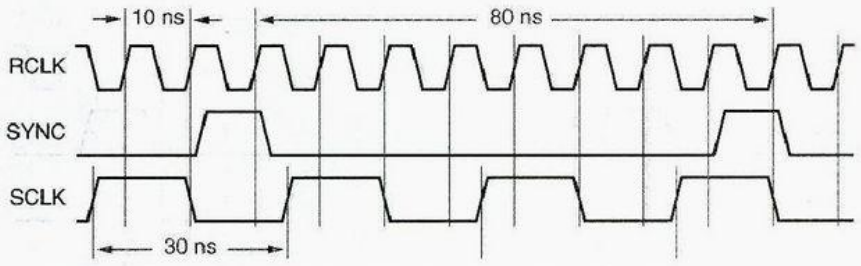


Figura 8-103
Enlace Ethernet y temporización del reloj del sistema.

bit a bit en un registro de corrimiento de 8 bits. Al mismo tiempo, un circuito de sincronización de byte busca patrones especiales en el flujo de datos recibido que indican fronteras de byte. Cuando se detecta uno de ellos, la señal SYNC se activa y se hace así en cada octavo subsiguiente tic de RCLK, de manera que SYNC es asertiva donde el registro de corrimiento contenga un byte alineado de 8 bits. El resto del sistema es temporizado por un reloj SCLK de 33.33 MHz. Necesitamos transferir cada byte alineado RBYTE[7:0] en un registro SREG en el dominio de SCLK. ¿Cómo podemos hacer esto?

La figura 8-103 ilustra una parte de la temporización. Vemos de inmediato que la señal alineada de byte, SYNC, es asertiva durante solamente 10 ns por byte. No tenemos esperanza de detectar de manera consistente esta señal con el SCLK asincrónico, cuyo periodo es más grande por unos 30 ns.

La estrategia casi universalmente seguida en este tipo de situaciones es transferir primero los datos alineados en un registro de retención HREG ("Holding Register") en el dominio del reloj *receptor* (RCLK). Esto nos da mucho tiempo más para ordenar las cosas: 80 ns en este caso. De este modo, el cuadro "?" en la figura 8-102 puede ser reemplazado por la figura 8-104, la cual muestra HREG y un cuadro marcado "SCTRL". El trabajo de SCTRL es afirmar SLOAD durante exactamente un periodo SCLK de 30 ns, de manera que la salida de HREG sea válida y estable para los tiempos de establecimiento y retención del registro SREG en el dominio SCLK. SLOAD sirve también como una señal de "nuevos datos disponibles" para el resto de la interfaz, indicando que un nuevo byte de datos aparecerá en SBYTE[7:0] durante el siguiente periodo SCLK.

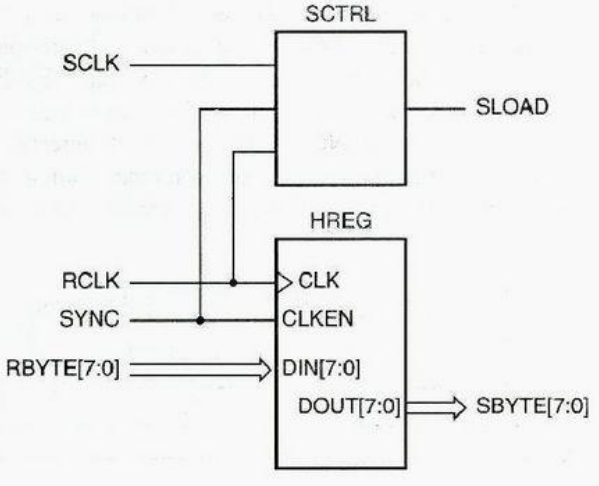


Figura 8-104
Control y registro de retención del byte.

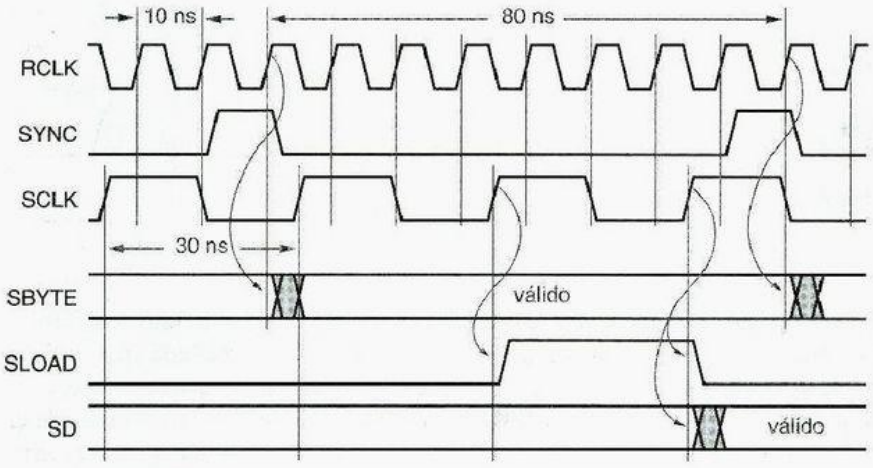


Figura 8-105
Temporización para SBYTE y temporización posible para SLOAD.

La figura 8-105 ilustra la posible temporización para SLOAD basada en este enfoque y el anterior diagrama de temporización.

La figura 8-106 es un circuito que puede generar SLOAD con la temporización deseada. La idea es utilizar SYNC para establecer un cerrojo S-R como un nuevo byte que se hace disponible. La salida de este cerrojo, NEWBYTE, es muestreada mediante FF1 en el dominio SCLK. Puesto que NEWBYTE no se encuentra sincronizada con SCLK, la salida SM de FF1 puede ser metaestable, pero no es empleada por FF2 hasta el nuevo tic de reloj, 30 ns más tarde. Suponiendo que la compuerta AND es razonablemente rápida, esto proporciona un abundante tiempo de resolución de metaestabilidad. La salida de FF2 es la señal SLOAD. La compuerta AND asegura que SLOAD es de solamente de una anchura de un periodo de SCLK; si SLOAD ya es 1, no puede ser establecida a 1 en el siguiente tic. Esto da tiempo para que el cerrojo S-R sea restablecido por SLOAD en preparación para el siguiente byte.

Un diagrama de temporización "típico" para todo el circuito se muestra en la figura 8-107. Puesto que SCLK es asincrónico con respecto a RCLK, puede tener una relación arbitraria con RCLK y SYNC. En la figura, hemos mostrado un caso donde la siguiente frontera ascendente SCLK se presenta bastante después de que NEWBYTE es establecido. Aunque la figura ilustra una ventana en la que SM y SM1 podrían ser metaestables en el caso general, la metaestabilidad no se presenta en realidad cuando la temporización aparece como se esquematiza. Posteriormente, mostraremos lo que puede pasar si el flanco SCLK se presenta cuando NEWBYTE está cambiando.

Deberíamos hacer diversas anotaciones acerca del circuito en la figura 8-106. En primer lugar, la señal SYNC debe estar libre de interferencias, puesto que controla la entrada S de un cerrojo, y debe ser suficientemente ancha para satisfacer el requerimiento de anchura de pulso mínima del cerrojo. Puesto que el cerrojo se establece en el flanco

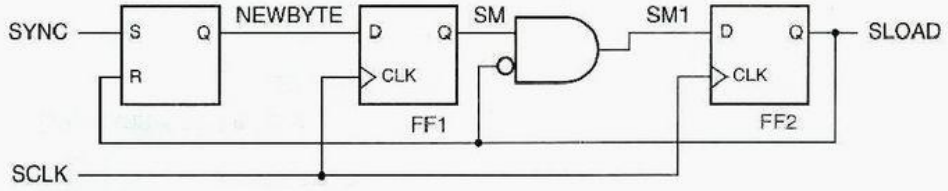


Figura 8-106
Circuito SCTRL para la generación de SLOAD.

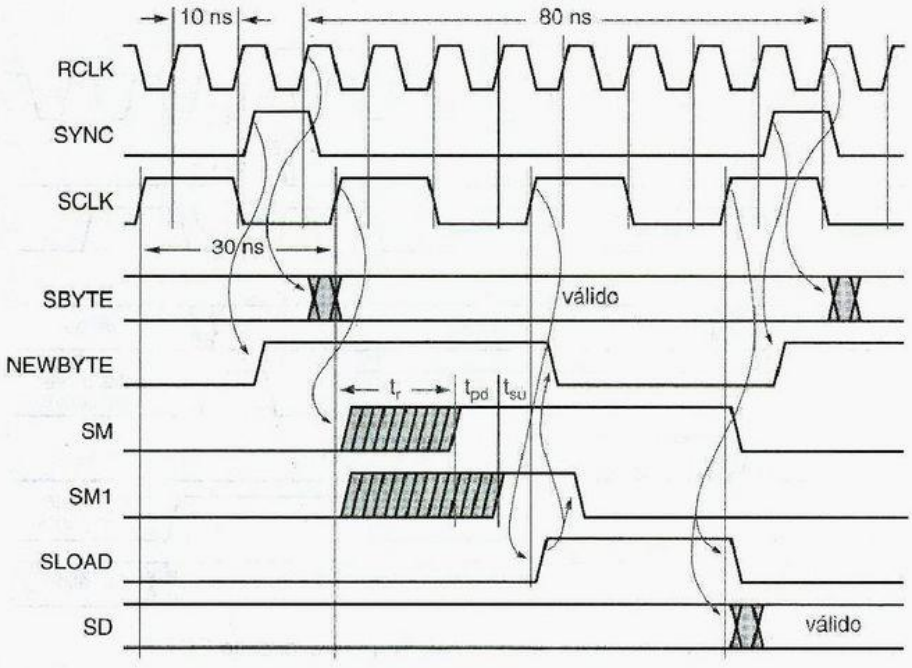


Figura 8-107
Temporización para el circuito SCTRL en la figura 8-106.

principal de SYNC, en realidad engañamos un poco; NEWBYTE puede afirmarse un poco *antes* de que un nuevo byte se encuentre disponible en realidad en HREG. Esto está bien, porque sabemos que toma dos periodos SCLK desde que NEWBYTE se muestrea hasta que se carga SREG. De hecho, podemos haber engañado aun más si una versión anterior de SYNC había estado disponible (vea el ejercicio 8.95).

Suponiendo que t_{su} es el tiempo de establecimiento de un flip-flop D y t_{pd} es el retardo de propagación ("propagation delay") de la compuerta AND en la figura 8-106, el tiempo de resolución de metaestabilidad disponible t_r es un periodo SCLK, 30 ns, menos $t_{su} + t_{pd}$, como se muestra en la figura 8-107. El diagrama de temporización también muestra por qué no podemos emplear SM directamente con la señal de restablecimiento para el latch S-R. Puesto que SM puede ser metaestable, podría descargar ruido. Por ejemplo, podría ser de magnitud semi-ALTO suficiente para reestablecer el latch pero posteriormente caer de vuelta a BAJO; en ese caso, SLOAD no podría llegar a establecerse y perderíamos un byte. Al utilizar en vez de la salida del sincronizador (SLOAD) tanto para el restablecimiento del latch como para la señal de carga en el dominio de SCLK, nos aseguramos que el nuevo byte sea detectado y manejado de manera consistente en ambos dominios de reloj.

La temporización que mostramos en la figura 8-107 es nominal, pero también tenemos que analizar lo que ocurre si SCLK tiene una relación de fase con RCLK y SYNC diferente de lo que se muestra. Usted debería ser capaz de convencerse a sí mismo que si se presenta la frontera SCLK al principio, de modo que muestre NEWBYTE justamente cuando va haciéndose ALTO, todo funcionará como antes, y la transferencia de datos sólo terminará un poco más pronto. El caso más interesante es cuando SCLK se presenta más tarde, de tal modo que sólo pierde NEWBYTE a medida que se va al nivel ALTO y lo captura un periodo SCLK más tarde. Esta temporización se ilustra en la figura 8-108.

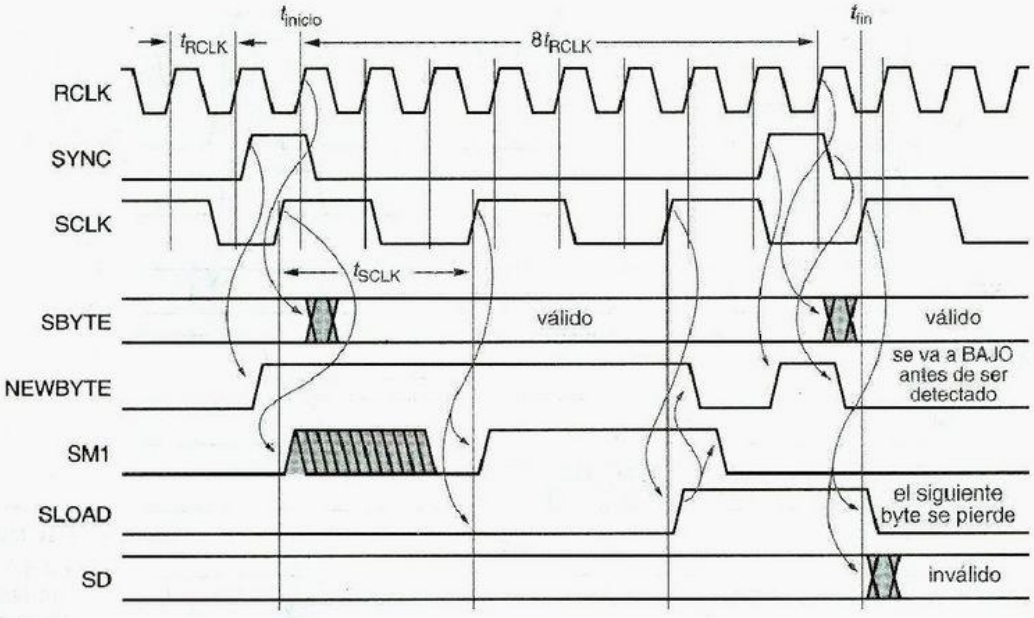


Figura 8-108 Temporización de retardo máximo para el circuito SCTRL.

En el diagrama de temporización, hemos mostrado a NEWBYTE elevándose aproximadamente al mismo tiempo que el flanco SCLK: menos que el t_{su} de FF1 antes del flanco. De esta forma, FF1 puede no ver NEWBYTE como ALTO o su salida puede llegar a ser metaestable, y no capturar sólidamente NEWBYTE hasta un periodo SCLK más tarde. Dos periodos SCLK después de eso, obtenemos la frontera SCLK que carga SBYTE en SREG.

Este escenario de temporización trae malas noticias, porque por el tiempo en que ocurre la carga, SBYTE ya se encuentra cambiando al *siguiente* byte recibido. Además, SLOAD se llega a afirmar durante, y un pequeño instante después del pulso SYNC para este siguiente byte recibido. Así, el latch tiene tanto S como R asertivas simultáneamente. Si son eliminados de manera simultánea, la salida del latch puede llegar a ser metaestable. Ahora bien, como hemos mostrado en el diagrama de temporización, si NEWBYTE(R) es negada al último, entonces el latch se deja en el estado de restablecimiento, y este siguiente byte recibido nunca es detectado y cargado en el dominio SCLK.

De esta forma, necesitamos analizar el caso de temporización de retardo máximo con cuidado para determinar si un sincronizador funcionará apropiadamente. La figura 8-108 muestra un punto de referencia de inicio t_{inicio} para el circuito SCTRL, específicamente el flanco RCLK en el cual se carga un byte en HREG, al final del pulso SYNC. El punto de referencia de terminación t_{fin} es el flanco SCLK en el cual SBYTE se carga en SREG. El retardo máximo entre estos dos puntos de referencia, al cual llamaremos t_{maxd}, es la suma de los siguientes componentes:

$-t_{RCLK}$ Menos un periodo RCLK, el retardo de t_{inicio} de vuelta al flanco en el cual SYNC fue asertiva. Este número es negativo porque SYNC es asertiva un tic de reloj antes del tic que en realidad carga HREG.

t_{CQ} Un retardo máximo flip-flop CLK a Q. Suponiendo que SYNC es una salida de flip-flop directa en el dominio de RCLK, éste es retardo desde la frontera de RCLK hasta que SYNC es asertiva.

t_{SQ} Retardo máximo desde S hasta Q en el latch S-R en la figura 8-106. Éste es el retardo para que NEWBYTE sea asertiva.

t_{su} Tiempo de establecimiento de FF1 en la figura 8-106. NEWBYTE debe ser asertiva en o antes del tiempo de establecimiento para garantizar la detección.

t_{SCLK} Un periodo SCLK. Puesto que RCLK y SCLK son asincrónicos, puede haber un retardo de hasta un periodo SCLK antes de que la siguiente frontera SCLK llegue también para muestrear NEWBYTE.

t_{SCLK} Después que NEWBYTE es detectado por FF1, SLOAD es asertiva en el siguiente tic de SCLK.

t_{SCLK} Después que SLOAD es asertiva, SBYTE se carga en SREG en el siguiente tic SCLK.

De esta forma, $t_{maxd} = 3t_{SCLK} + t_{CQ} + t_{SQ} + t_{su} - t_{RCLK}$. Se deben definir otros parámetros para terminar el análisis:

t_h El tiempo de retención de SREG.

$t_{CQ(min)}$ El mínimo retardo de CLK a Q de HREG, suponiéndolo conservadoramente como de valor 0.

t_{rec} El tiempo de recuperación del latch S-R, el tiempo mínimo permitido entre S negada y R negada (vea el cuadro en la página 537).

Para cargarse con éxito en SREG, SBYTE debe permanecer válido hasta por lo menos el tiempo $t_{fin} + t_h$. El punto en el que SBYTE cambia y se hace inválido es 8 periodos de RCLK después de t_{inicio}, más $t_{CQ(min)}$. De este modo, para un funcionamiento apropiado del circuito, debemos tener

$$t_{fin} + t_h \leq t_{inicio} + 8t_{RCLK}$$

Para el caso de retardo máximo, sustituimos $t_{fin} = t_{inicio} + t_{maxd}$ en esta relación y restamos t_{inicio} de ambos lados para obtener

$$t_{maxd} + t_h \leq 8t_{RCLK}$$

Sustituyendo el valor de t_{maxd} y reacomodando términos, obtenemos

$$3t_{SCLK} + t_{CQ} + t_{SQ} + t_{su} + t_h \leq 9t_{RCLK} \qquad (8\text{-}1)$$

como requerimiento para un funcionamiento correcto del circuito, lo cual es demasiado malo. Incluso si supusiéramos retardos de componentes muy cortos (t_{CQ}, t_{SQ}, t_{su}, t_h), sabemos que $3t_{SCLK}$ (90 ns) más cualquier cosa va a ser de más de $9t_{RCLK}$ (también 90 ns). De manera que este diseño jamás funcionará apropiadamente en el caso de retardo máximo.

Aun si el análisis de carga-retardo diera un buen resultado, tendríamos todavía que considerar los requerimientos para un funcionamiento apropiado del propio circuito SCTRL. En particular, debemos asegurar que cuando el pulso SYNC se presente para el

siguiente byte, no es negado hasta el tiempo t_{rec} después de que SLOAD para el byte anterior fue negado. Así, otra condición para tener un funcionamiento apropiado es

$$t_{fin} + t_{CQ} + t_{rec} \leq t_{inicio} + 8t_{RCLK} + t_{CQ(min)}$$

Sustituyendo y simplificando como antes, obtenemos otro requerimiento que no es satisfecho por nuestro diseño:

$$3t_{SCLK} + 2t_{CQ} + t_{SQ} + t_{su} + t_{rec} \leq 9t_{RCLK} \tag{8-2}$$

Existen varias maneras en que podemos modificar nuestro diseño para satisfacer los requerimientos de temporización del peor de los casos. Al principio de nuestra discusión, advertimos que "hicimos trampa" al afirmar SYNC un periodo RCLK antes que los datos en HREG sean válidos, y que en realidad podemos salir impunes haciendo a SYNC asertivo incluso antes. Hacer esto puede ayudarnos a satisfacer el requerimiento de retardo máximo, porque reduce el término "$8t_{RCLK}$" en el lado derecho de las relaciones. Por ejemplo, si afirmamos SYNC dos periodos RCLK antes, reduciríamos este término a "$6t_{RCLK}$". Sin embargo, no hay nada gratis; no podemos hacer asertiva de manera arbitraria a SYNC antes. Debemos considerar también un caso de *retardo mínimo*, para asegurar que el nuevo byte está en realidad disponible en HREG cuando SBYTE se carga en SREG. El retardo mínimo t_{maxd} entre t_{inicio} y t_{fin} es la suma de los siguientes componentes:

$-nt_{RCLK}$ Menos n periodos RCLK, el retardo desde t_{inicio} de vuelta a la frontera en el cual SYNC fue asertiva. En el diseño original, $n = 1$.

$t_{CQ(min)}$ Éste es el mínimo retardo desde la frontera de RCLK hasta que SYNC es asertiva, suponiendo de manera conservadora que es 0.

t_{SQ} Éste es el retardo para que NEWBYTE sea asertiva, suponiendo de nuevo que sea 0.

$-t_h$ Menos el tiempo de retención de FF1 en la figura 8-106. NEWBYTE puede ser asertiva al final del tiempo de retención y todavía ser detectada.

$0t_{SCLK}$ Cero veces el periodo SCLK. Podemos ser "afortunados" y tener la frontera SCLK llegando justo cuando el tiempo de retención de FF1 se está finalizando.

t_{SCLK} Un retardo de un periodo SCLK para afirmar SLOAD, como antes.

t_{SCLK} Un retardo de un periodo SCLK para cargar SBYTE en SREG, como antes.

En otras palabras, $t_{mind} = 2t_{SCLK} - t_h - nt_{RCLK}$.

Para este caso, debemos asegurarnos que el nuevo byte se ha propagado hacia la salida de HREG cuando la ventana de tiempo de establecimiento de SREG comienza, de manera que debemos tener

$$t_{fin} - t_{su} \geq t_{inicio} + t_{co},$$

donde t_{co} es el máximo retardo de reloj a salida ("clock-to-output") de HREG. Sustituyendo $t_{fin} = t_{inicio} + t_{mind}$ y restando t_{inicio} de ambos lados, obtenemos

$$t_{mind} - t_{su} \geq t_{co}.$$

Sustituyendo el valor de t_{mind} y reacomodando, obtenemos el término final,

$$2t_{SCLK} - t_h - t_{su} - t_{co} \geq nt_{RCLK} \tag{8-3}$$

Si, por ejemplo, t_h, t_{su} y t_{co} son cada uno de 10 ns, el máximo valor de n es 3; no podemos generar SYNC más de dos tics de reloj antes de su posición original en la figura 8-108. Esto puede o no puede ser suficiente para resolver el problema de retardo máximo, dependiendo de otros valores de retardo; esto se explora para un conjunto particular de componentes en el ejercicio 8.95.

Moviendo el pulso SYNC antes puede no proporcionar suficiente mejora del retardo o puede no ser una opción disponible en algunos sistemas. Una solución alternativa consiste en aumentar el tiempo entre las transferencias sucesivas de datos desde un dominio de reloj al otro. Siempre podemos hacer esto porque se pueden transferir más bits por sincronización. En el ejemplo de interfaz Ethernet podríamos recolectar 16 bits a la vez en el dominio RCLK y transferir 16 bits a la vez al dominio SCLK. Esto cambia los términos de $8t_{RCLK}$ previamente establecido a 16 t_{RCLK}, proporcionando un mayor margen para los requerimientos de temporización con retardo máximo. Una vez que se han transferido 16 bits en el dominio SCLK, podemos aún dividirlos en dos segmentos de 8 bits si necesitamos procesar los datos, un byte a la vez.

También puede ser posible mejorar el desempeño mediante la modificación del diseño del circuito SCTRL. La figura 8-109 muestra una versión donde SLOAD es generado directamente por el flip-flop que muestrea NEWBYTE. En esta forma, SLOAD aparece un periodo SCLK más pronto que en nuestro circuito SCTRL original. También, el latch S-R se borra más pronto. Este circuito funciona solamente si un par de suposiciones de clave son verdaderas:

1. Un tiempo de resolución de metaestabilidad reducido para FF1 es aceptable, igual al tiempo en que SCLK está en ALTO. La metaestabilidad debe resolverse antes que SCLK se vaya a nivel BAJO, porque en ese momento se borra el latch S-R si SLOAD está en ALTO.
2. El tiempo de establecimiento de la entrada CLKEN de SREG (figura 8-102) es menor que o igual al tiempo en que SCLK está en BAJO. Bajo la suposición anterior, la señal SLOAD aplicada a CLKEN puede ser metaestable hasta que SCLK se vaya al nivel BAJO.
3. El tiempo en que SCLK está en BAJO es suficientemente largo para generar un pulso de restablecimiento en RNEW que satisface el requerimiento de anchura de pulso mínimo del latch S-R.

Note que estos comportamientos hacen la operación apropiada del circuito dependiente del ciclo de trabajo de SCLK. Si SCLK es relativamente lenta y su ciclo de trabajo está cercano al 50%, este circuito generalmente funcionará bien. Pero si SCLK es demasiado rápido o tiene un ciclo de trabajo muy pequeño, muy grande o impredecible, debe utilizarse el enfoque del circuito original.

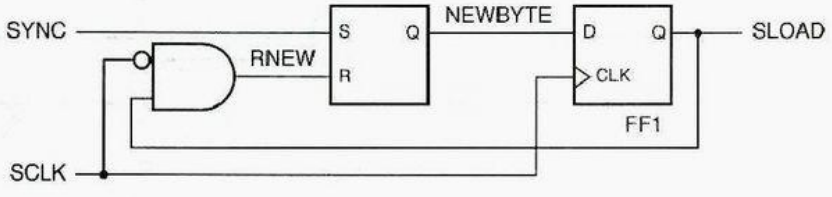


Figura 8-109
Circuito SCTRL de medio periodo de reloj para generar SLOAD.

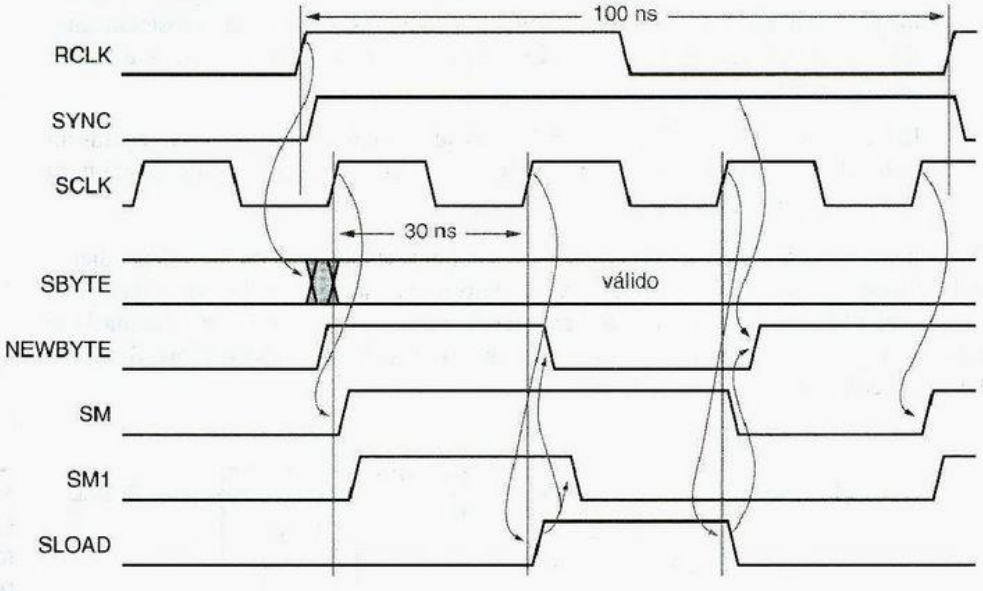
Todos estos esquemas de sincronización requieren que las frecuencias de reloj se encuentren dentro de un cierto intervalo entre sí para un funcionamiento adecuado del circuito. Esto debe ser considerado para pruebas, donde los relojes por lo regular se encuentran corriendo más lentamente, y para mejoras, donde uno o ambos relojes pueden correr más rápido. Por ejemplo, en el ejemplo de la interfaz Ethernet, no podríamos cambiar la frecuencia del Ethernet estándar de 100 Mbps, pero podríamos mejorar el bus PCI de 33 a 66 MHz.

Los problemas causados por los cambios de frecuencia de reloj pueden ser sutiles. Para obtener un mejor manejo de lo que puede ir mal, es útil considerar cómo trabaja (¡o cómo no trabaja!) un sincronizador si una frecuencia de reloj cambia por un factor de 10 o más.

Por ejemplo, ¿qué ocurre a la temporización del sincronizador en la figura 8-107 si cambiamos RCLK de 100 MHZ a 10 MHz? A primera vista, parecería que todo está bien, puesto que ahora llega un byte solamente una vez cada 800 ns, dando mucho más tiempo para que el byte sea transferido en el dominio SCLK. Desde luego, la ecuación (8-1) y la ecuación (8-2) se satisfacen con mucho más margen. Sin embargo, la ecuación (8-3) ya no se satisface, ¡a menos que reduzcamos n a cero! Esto podría llevarse a cabo mediante la generación de SYNC un tic de RCLK más tarde de lo que se muestra en la figura 8-107.

Pero incluso con este cambio, *todavía* existe un problema. La figura 8-110 muestra la nueva temporización, incluyendo el último pulso SYNC. El problema en que el pulso SYNC dura ahora 100 ns. Como antes, NEWBYTE (la salida del latch S-R en la figura 8-106) es establecida por SYNC y es borrada por SLOAD. El problema es que

Figura 8-110 Temporización de sincronizador con RCLK lento (10 MHz).



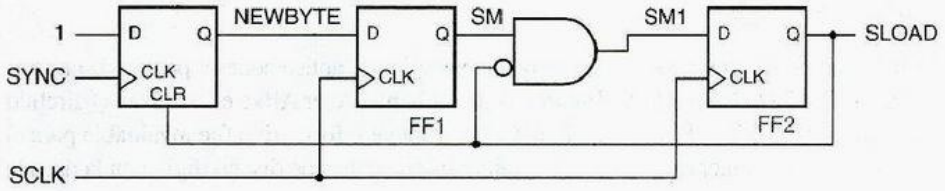


Figura 8-111
Sincronizador con detección SYNC disparada por flanco.

cuando SLOAD se retira, SYNC todavía es asertiva, como se muestra en el nuevo diagrama de temporización. De este modo, ¡el nuevo byte será detectado y transferido dos veces!

La solución al problema consiste en detectar solamente la frontera principal de SYNC, de manera que el circuito no sea sensible a la longitud del pulso SYNC. Una manera común de hacer esto es mediante el reemplazo del latch S-R con un flip-flop D disparado por flanco, como se ilustra en la figura 8-111. El flanco principal de SYNC establece el flip-flop, mientras que SLOAD se utiliza como un borrado asincrónico como antes.

El circuito en la figura 8-111 resuelve el problema del RCLK lento, pero también cambia la obtención de las ecuaciones 8-1 a la 8-3 y puede hacer la temporización más restringida en algunas áreas (vea el ejercicio 8.96). Otra desventaja es que este circuito no puede ser realizado en un PLD típico, que tiene todos los flip-flops controlados por el mismo reloj; en su lugar, debe emplearse un flip-flop discreto para detectar SYNC.

Después de leer casi diez páginas para analizar solamente un ejemplo "simple", usted debería tener una idea de las dificultades que se encuentran para tener un diseño de circuito de sincronización correcto. Varias instrucciones pueden ser de gran ayuda:

- Minimice el número de diferentes dominios de reloj en un sistema.
- Identifique con claridad todas las fronteras de reloj y proporcione sincronizadores claramente identificados en esas fronteras.
- Proporcione suficiente tiempo de resolución de metaestabilidad para cada sincronizador de manera que sean raras las fallas de sincronizador, mucho menos probables que otras fallas de hardware.
- Analice el comportamiento del sincronizador sobre un intervalo de escenarios de temporización, incluyendo relojes más rápidos y más lentos que se pueden aplicar como resultado de las verificaciones o mejoras del sistema.
- Simule también el comportamiento del sistema sobre una amplia gama de escenarios de temporización.

Esta última instrucción es una trampa para los diseñadores de sistemas digitales modernos, quienes habitualmente confían en simuladores sofisticados de lógica de alta velocidad para hallar sus errores. Pero eso no es un sustituto para entender las primeras cuatro directrices. Ignorarlas puede llevar a problemas que no se pueden detectar mediante un pequeño y típico número de escenarios de simulación. ¡De todos los circuitos digitales, los sincronizadores son los únicos para los cuales es sumamente importante tener "el diseño correcto"!

Referencias

Probablemente el primer conjunto general de ejemplos de aplicaciones y partes MSI aparecieron en *The TTL Applications Handbook*, editado por Peter Alfke e Ib Larsen (Fairchild Semiconductor, 1973). Este libro altamente práctico e informativo fue invaluable para el autor y para otros muchos quienes desarrollamos currículas de diseño digital en la década de 1970.

Otro libro que enfatiza el diseño con funciones lógicas secuenciales y combinacional en gran escala es el *Digital Design with Standard MSI and LSI*, segunda edición, de Thomas R. Blakeslee (Wiley, 1979). La cobertura de Blakeslee sobre el concepto de compromisos espacio/tiempo es excelente, y también es uno de los primeros en presentar al microprocesador como un "circuito lógico universal".

Una excelente fuente de información acerca de prácticas de diseño digital es los sitios web de los fabricantes, navegando desde los casi olvidados hasta los que están por descubrirse todavía. Por ejemplo, una amplia discusión de circuitos de retención de bus puede encontrarse en el artículo "Implications of Slow or Floating CMOS Inputs" de Texas Instruments (publicación SCBA004B, diciembre de 1997), disponible en el sitio web de TI ("Texas Instruments") en `www.ti.com`. Otra discusión aparece en "Designing with Bushold" (*sic*, publicación AN-5006, abril de 1999) de Fairchild Semiconductor, en `www.fairchildsemi.com`.

Los anuncios y hojas de datos para todas las clases de nuevas y mejoradas partes MSI y más grandes también pueden encontrarse en la red de internet. Siguiendo la afirmación de cierto fabricante de automóviles en los años sesenta y posteriormente a finales de los noventa, de que "entre más ancho, mejor", los fabricantes lógicos también han introducido registros, controladores y transceptores de "bus amplio" que meten a la fuerza 16, 18 o incluso 32 bits de función en un encapsulado de montaje superficial con alto número de terminales. Las descripciones de muchas partes de esta clase pueden hallarse en el sitio Web de Texas Instruments (busque por "widebus", o sea, bus amplio). Otros sitios con gran variedad de hojas de datos lógicas, notas de aplicación e información adicional incluyen a Motorola (`www.mot.com`), Fairchild Semiconductor (`www.fairchildsemi.com`) y Philips Semiconductor (`www.philipslogic.com`).

El campo del diseño lógico está cambiando muy rápidamente, de tal modo que en ocasiones me gustaría escribir ficción, de modo que no tendría que revisar las discusiones "prácticas" en este libro cada que pasan unos cuantos años. Afortunadamente para mí, este libro cubre algunos temas teóricos inmutables (alias "principios") y este capítulo no es la excepción. Los riesgos lógicos han sido conocidos desde por lo menos la década de 1950, y los riesgos funcionales fueron discutidos por Edward J. McCluskey en *Logic Design Principles* (Prentice-Hall, 1986). Los campos de Galois fueron inventados hace algunos siglos y sus aplicaciones a códigos de corrección de errores, así como también a los contadores LFSR de este capítulo, se describen en libros introductorios sobre teoría de codificación, incluyendo *Error-Control Techniques for Digital Communication* por A. M. Michelson y A. H. Levesque (Wiley-Interscience, 1985). Una teoría matemática de la descomposición de la máquina de estado ha sido estudiada por años; Zvi Kohavi dedica un capítulo al tema en su clásico libro *Switching and Finite Automata Theory*, segunda edición (McGraw-Hill, 1978). Pero ahora volvamos a lo menos esotérico.

Como mencionamos en la sección 8.4.5, algunos PLD y CPLD contienen estructuras XOR que permiten diseñar contadores grandes sin un gran número de términos de producto. Esto requiere un conocimiento algo más profundo de ecuaciones de excitación del contador, como se describe en la sección 10.5 de la segunda edición de este libro. Este material también puede encontrarse en la red de internet en el sitio www.ddpp.com.

Los temas generales de sesgo de reloj y temporización multifásica se discuten en los *Logic Design Principles* de McCluskey, mientras que una discusión ilustrativa de estos temas como se aplican a los circuitos VLSI puede encontrarse en *Introduction to VLSI Systems* por Carver Mead y Lynn Conway (Addison-Wesley, 1980). Mead y Conway también proporcionan una introducción al importante tema de los *sistemas autotemporizados* que eliminan el reloj del sistema, permitiendo que cada elemento lógico proceda a su propia velocidad. Para dar honor a quien honor merece, advertimos que todo el material de Mead y Conway acerca de temporización de sistemas, incluyendo una extraordinaria discusión sobre metaestabilidad, aparece en su capítulo 7 escrito por Charles L. Seitz.

sistemas autotemporizados

Thomas J. Chaney pasó décadas estudiando y haciendo informes acerca del problema de la metaestabilidad. Uno de sus más importantes trabajos, "Measured Flip-Flop Responses to Marginal Triggering" (*IEEE Trans. Comput.*, Vol. C-32, No. 12, diciembre de 1983, págs. 1207-1209), informa algunos de los resultados que exhibimos en la tabla 8-35.

Para los interesados en las matemáticas, Lindsay Kleeman y Antonio Cantoni han escrito "On the Unavoidability of Metastable Behavior in Digital Systems" (*IEEE Trans. Comput.*, Vol. C-36, No. 1, Enero de 1987, págs. 109-112); el título lo dice todo. Los mismos autores postulan la pregunta "Can Redundancy and Masking Improve the Performance of Synchronizers?" (*IEEE Trans. Comput.*, Vol. C-35, No. 7, julio de 1986, págs. 643-646). Su respuesta en ese artículo era "No", pero una respuesta de un revisor provocó que cambiaran sus opiniones a "Quizás". ¡Obviamente, ellos mismos se habían hecho metaestables! Tener dos autores y un revisor no mejoró su desempeño, de modo que la respuesta obvia a su pregunta original es "¡No!" En cualquier caso, los artículos de Kleeman y Antonio proporcionan un buen conjunto de indicadores para encauzar las referencias eruditas sobre la metaestabilidad.

El conjunto más amplio de referencias iniciales acerca de la metaestabilidad (sin incluir a los filósofos griegos o Devo) es "A Guide Tour of 35 Years of Metastability Research" de Martin Bolton (*Proc. Wescon 1987*, Session 16, "Everything You Might Be Afraid to Know about Metastability", Wescon Session Records, www.wescon.com, 8110 Airport Blvd., Los Angeles, CA 90045).

En años recientes, a medida que las velocidades de reloj del sistema se han incrementado constantemente, muchos fabricantes de CI han llegado a ser mucho más conscientes acerca de la medición y publicación de las características de metaestabilidad de sus dispositivos, con frecuencia la compañía Texas Instruments (www.ti.com) proporciona una muy buena discusión que incluye circuitos de prueba y parámetros de medida para diez diferentes familias lógicas en "Metastable Response in 5-V Logic Circuits" por Eilhard Haseloff (TI pub. SDYA006, 1997). Cypress Semiconductor (www.cypress.com) publica una nota de aplicación, "Are Your PLDs Metastable?" (1997) que es una excelente referencia incluyendo algo de historia (¡de regreso hasta

1952!), un análisis de circuito analógico de la metaestabilidad, circuitos de prueba y medición, y parámetros de metaestabilidad para los PLD de Cypress. Otra nota reciente es "Metastability Considerations" de Xilinx Corporation (`www.xilinx.com`, publ. XAPP077, 1997), la que ofrece parámetros de medición para sus familias XC7300 y XC9500 de CPLD. De particular interés es la metodología y circuito inteligente que les permiten contar eventos metaestables *dentro* del dispositivo, aun cuando las formas de onda metaestables no sean observables en las terminales externas.

La mayoría de los textos de diseño digital proporcionan una buena cobertura de la metaestabilidad, impulsada por la existencia de la metaestabilidad en los circuitos reales y quizás también por la competencia: desde 1990, el libro de texto que usted está leyendo ha sido promotor del tema al introducir la metaestabilidad en su cobertura inicial de los circuitos secuenciales. Por el lado analógico del asunto, Howard Johnson y Martin Graham proporcionan una agradable introducción y una descripción de cómo observar los estados metaestables en su *High-Speed Digital Design: A Handbook of Black Magic* (Prentice-Hall, 1993).

Problemas propuestos

8.1 Supongamos que en la tabla 8-2, el segundo banco de RAM (`RAMCS1`) es decodificado en lugar de utilizar la expresión `( (ABUS >= 0x010) & (ABUS < 0x020) )`. ¿Esto produce los mismos resultados que la expresión original, `(ABUS == RAMBANK0)`? Explique por qué.

8.2 Determine el número de fusibles en cada uno de los dispositivos PAL de la tabla 8-9.

8.3 ¿Cuántos fusibles están contenidos en el 16V8 como se describe en el texto? (El dispositivo comercial tiene fusibles adicionales, no descritos en el texto, para seguridad e información específica del usuario).

8.4 ¿Cuántos fusibles están contenidos en el 22V10 como se describe en el texto? (El dispositivo comercial tiene fusibles adicionales, no descritos en el texto, para seguridad e información específica del usuario).

8.5 Determine f_{max} con retroalimentación externa para todos los dispositivos en la tabla 8-10.

8.6 Determine f_{max} con retroalimentación interna para todos los dispositivos en la tabla 8-10.

8.7 Escriba un programa ABEL para un 16V8 que ofrezca exactamente la misma funcionalidad que la de un 74x374.

8.8 Escriba un programa ABEL para un GAL16V8 o GAL20V8 que ofrezca exactamente la misma funcionalidad que la de un 74x377.

8.9 Compare los retardos de programación de AVALID para una salida de selección de chip para los dos enfoques de decodificación mostrados en las figuras 8-15 y 8-16. Suponga que se utilizan latchs 74FCT373 y dispositivos 16V8C de 10 ns en ambos diseños. Repita para el retardo de ABUS para una salida de selección de chip.

8.10 ¿Qué podría ocurrir si se reemplazan los flip-flops (dispositivos biestables) D disparados por frontera en la figura 7-38 con latchs D?

8.11 Modifique el programa ABEL de la tabla 8-12 para realizar la función de un contador de décadas 74x162.

8.12 Modifique el programa ABEL en la tabla 8-12 para realizar la función de un contador hacia arriba/hacia abajo 74x169. ¿Todavía cabrá en un 16V8?

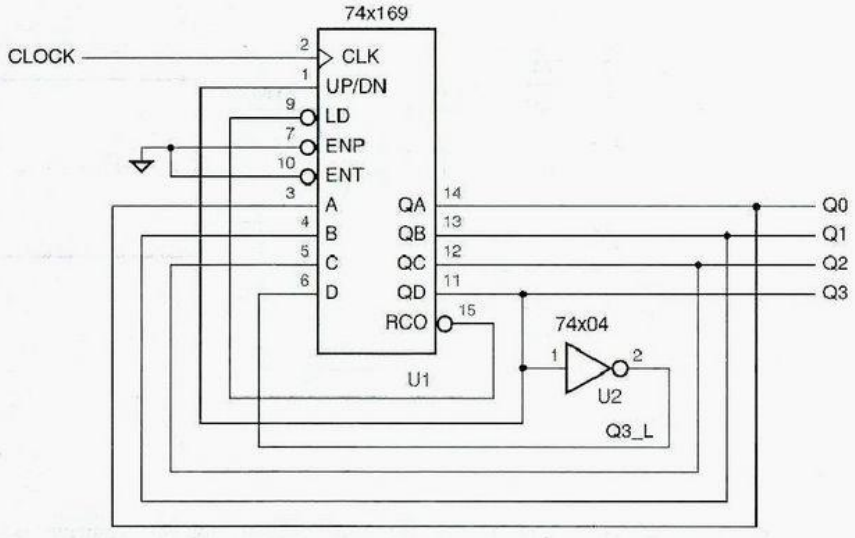


Figura X8.13

8.13 ¿Cuál es la secuencia de conteo del circuito ilustrado en la figura X8-13?

8.14 ¿Cuál es el comportamiento del circuito contador de la figura 8-36 si se construye a partir de un 74x161 en lugar de un 74x163?

8.15 ¿Cuál es el comportamiento del circuito contador de la figura 8-36 si la entrada inferior de U2 se encuentra conectada a Q0 en vez de Q1?

8.16 Un contador 74x163 se conecta con entradas ENP, ENT, A y D siempre en nivel ALTO, las entradas B y C siempre en nivel BAJO, entrada LD_L = (QB · QC)' y entrada CLR_L = (QC · QD)'. La entrada CLK es conectada a una señal de reloj libre de carrera. Dibuje un diagrama lógico para este circuito. Suponiendo que el contador arranque en el estado 0000, escriba la secuencia de salida en QD QC QB QA para los siguientes 15 tics de reloj.

8.17 Determine las anchuras de las interferencias mostradas en la figura 8-43 en la salida Y2_L de un decodificador 74x138, suponiendo que el '138 se encuentra internamente estructurado como se muestra en la figura 5-37(a) y que cada compuerta interna tiene un retardo de 10 ns.

8.18 Comenzando con el estado 0001, escriba la secuencia de estados para un contador LFSR de 4 bits diseñado de acuerdo a la figura 8-68 y la tabla 8-21.

8.19 Calcule el mínimo periodo de reloj de la unidad de datos en la figura 8-82. Utilice los retardos de propagación máximos dados en la tabla 5-3 para partes combinacionales LS-TTL. A menos que usted pueda obtener los números reales de un libro de datos de TTL, suponga que todos los registros requieren de un tiempo de establecimiento mínimo de 10 ns en las entradas y que tienen un retardo de propagación de 20 ns desde el reloj hasta las salidas. Indique cualquier suposición que haga sobre los retardos en la unidad de control.

8.20 Calcule el MTBF de un sincronizador que se construye de acuerdo a la figura 8-96 y utiliza 74F74s, suponiendo una frecuencia de reloj de 25 MHz y una velocidad de transición asíncrona de 1 MHz. Suponga que el tiempo de establecimiento de un 'F74 es de 5 ns y que el tiempo de retención es cero.

8.21 Calcule el MTBF del sincronizador mostrado en la figura X8.21, suponiendo una frecuencia de reloj de 25 MHz y una velocidad de transición asincrónica de 1 MHz. Suponga que el tiempo de establecimiento $t_{\text{establecimiento}}$ y el retardo de propagación t_{pd} desde el reloj hasta Q o QN en un 74ALS74 son ambos de 10 ns.

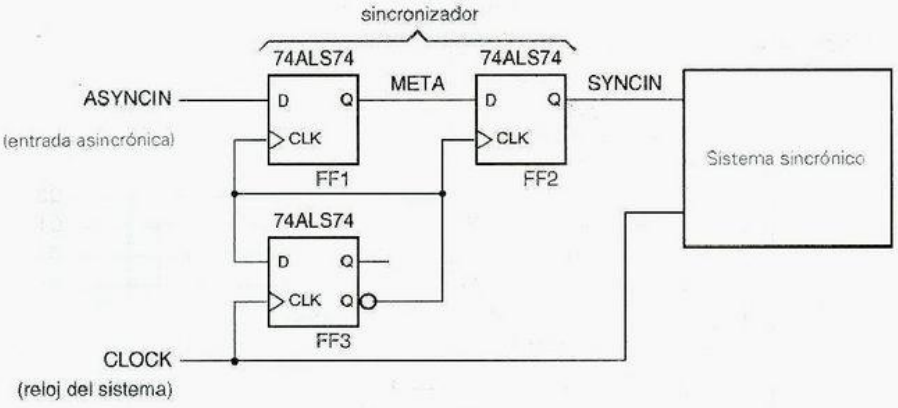


Figura X8.21

Ejercicios

8.22 ¿Qué tiene que decir el TTL Data Book sobre poner en corto momentáneamente las salidas de una compuerta a tierra como lo hacemos en el circuito de inhibición de rebote de interruptor de la figura 8-5?

8.23 Investigue el comportamiento del circuito de inhibición de rebote de interruptor de la figura 8-5 si se utilizan inversores 74HCT04; repita lo anterior para inversores 74AC04.

8.24 Suponga que se le solicita diseñar un circuito que produzca una entrada lógica sin rebote desde un interruptor SPST (single-pole, *single* throw). ¿Con qué problema inherente se enfrentará usted?

8.25 Explique por qué los circuitos de bus de retención CMOS no trabajan bien en buses de tres estados con dispositivos TTL conectados. (*Sugerencia:* considere las características de entrada TTL).

8.26 Escriba un programa VHDL simple que combine en latch la dirección y el decodificador de latch de la figura 8-16 y de la tabla 8-2. Haga uso del nombre de señal LA[19:0] para las salidas de dirección del latch.

8.27 Diseñe un contador iterativo de 4 bits que utilice cuatro flip-flops (dispositivos biestables) D y ningún otro componente adicional.

8.28 ¿Cuál es el máximo retardo de propagación desde reloj hasta la salida pare el contador iterativo de 4 bits del ejercicio 8.27 que utiliza flip-flops (dispositivos biestables) 74HCT? Repita el ejercicio, empleando flip-flops (dispositivos biestables) 74AHCT y 74LS74.

8.29 Diseñe un contador iterativo *hacia abajo* o *descendente* de 4 bits con cuatro flip-flops (dispositivos biestables) D y ningún otro componente adicional.

8.30 ¿Qué limita la máxima velocidad de conteo de un contador iterativo, si usted no insiste en poder leer el valor del contador todas las veces?

8.31 Basado en el enfoque de diseño del ejercicio 8.27 y la respuesta del ejercicio 8.30, ¿cuál es la máxima velocidad de conteo (frecuencia) para un contador iterativo de 4 bits que utiliza flip-flops (dispositivos biestables) 74HCT? Repita el ejercicio, utilizando flip-flops (dispositivos biestables) 74AHCT y 74LS74.

8.32 Escriba una fórmula para la máxima frecuencia de reloj del circuito contador binario serie sincrónico en la figura 8-28. En su fórmula, haga que t_{TQ} denote el retardo de propagación desde T hasta Q en un flip-flop (dispositivo biestable) T, $t_{establecimiento}$ el tiempo de establecimiento de la entrada EN hasta la frontera ascendente de T, y t_{AND} el retardo de una compuerta AND.

8.33 Repita el ejercicio 8.32 para el circuito contador binario paralelo sincrónico en la figura 8-29, y compare los resultados.

8.34 Repita el ejercicio 8.32 para un contador binario serie sincrónico de n bits.

8.35 Repita el ejercicio 8.32 para un contador binario paralelo sincrónico de n bits. ¿Más allá de qué valor de n deja de ser válida o en su fórmula?

8.36 Con la ayuda de un contador binario de 4 bits 74x163, diseñe un circuito contador modulo 11 con la secuencia de conteo 3, 4, 5, ..., 12, , 13, 3, 4, ...

8.37 Busque el diagrama lógico interno para un contador de décadas sincrónico 74x162 en un libro de datos, y escriba su tabla de estado en el estilo de la tabla 8-11, incluyendo su comportamiento de conteo en los estados sin utilizar 10-15.

8.38 Elabore un esquema en cascada para el 74x163, de manera análoga a la estructura del contador en paralelo sincrónico de la figura 8-29, de tal modo que la velocidad de conteo máxima sea la misma para cualquier contador con hasta 36 bits (nueve '163). Determine la máxima velocidad de conteo, utilizando las especificaciones de retardo en el peor de los casos, de la hoja de especificaciones técnicas del fabricante, para los dispositivos número '163 y cualquier componente SSI que se utiliza en la conexión cascada.

8.39 Diseñe un contador módulo 129 empleando dos 74x163 y un inversor simple.

8.40 Escriba un programa ABEL para un contador módulo N de 8 bits con entrada de carga que utiliza un PAL22V10, donde el valor de N está especificado por una constante N en el programa.

8.41 Diseñe un circuito sincrónico temporizado con cuatro entradas, N3, N2, N1 y N0, que representen un entero N en el intervalo 0-15. El circuito tiene una salida simple Z que se asertiva para exactamente N tics de reloj durante cualquier intervalo de 16 tics (suponiendo que N se mantiene constante durante el intervalo de observación). (*Sugerencia:* utilice lógica combinacional con un 74x163 establecido como un contador de división entre 16 libre de carrera. Los tics en los cuales Z es asertiva deberían espaciarse tan equidistantes como sea posible, es decir, el segundo tic cuando $N = 8$, el cuarto cuando $N = 4$, y así sucesivamente.)

8.42 Modifique el circuito del ejercicio 8.41 de manera que Z produzca N *transiciones* en cada intervalo de 16 tics. El circuito resultante se conoce como un *multiplicador de velocidad binaria* y alguna vez fue vendido como una parte TTL MSI, la 7497. (*Sugerencia:* Dispare el reloj con la salida de nivel del circuito anterior.)

multiplicador de velocidad

8.43 Repita los ejercicios 8.41 y 8.42 usando una entrada de 8 bits N7..N0, y realice el circuito empleando un programa ABEL para un simple PAL22V10.

8.44 Repita los ejercicios 8.41 y 8.42 utilizando una entrada de 8 bits N7..N0, y describa el diseño con la ayuda de un programa VHDL de comportamiento.

8.45 A un diseñador digital (¡el autor!) se le solicitó en el último minuto agregar nueva funcionalidad a un PCB que tenía lugar para solamente un CI MSI de 16 terminales más. El PCB ya tenía una señal de reloj de 16 MHz, MCLK, y una señal de selección de reserva, controlada por microprocesador SEL. Se había pedido al diseñador que proporcionara una nueva señal de reloj, UCLK, cuya frecuencia sería de 8 MHz o 4 MHz dependiendo del valor de SEL. Para poner peores las cosas, el PCB no tenía compuertas SSI de reserva, y se requería que el UCLK tuviera un ciclo de trabajo de 50% para ambas frecuencias. Le tomó al diseñador aproximadamente cinco minutos proponer un circuito. Ahora es su turno para hacer lo mismo. (*Sugerencia:* el diseñador ya había considerado que el 74x163 sería el bloque de construcción fundamental del diseño de circuito secuencial truncado.)

8.46 Diseñe un contador módulo 16, utilice un 74x169 y como máximo un encapsulado SSI, con la siguiente secuencia de conteo: 7, 6, 5, 4, 3, 2, 1, 0, 8, 9, 10, 11, 12, 13, 14, 15, 7, ...

8.47 Escriba un programa ABEL para un contador de 8 bits que realice una secuencia de conteo similar a la del ejercicio 8.46.

8.48 Diseñe un contador binario ascendente/descendente para controlar el elevador de un edificio de 20 pisos, mediante un 16V8 simple. El contador debería tener entradas de habilitación y de control ascendente/descendente. Debería enclavarse en el estado 1 cuando se cuente hacia abajo, enclavarse en el estado 21 cuando se cuente hacia arriba, y saltar el estado 13 en cualquiera de los modos. Dibuje un diagrama lógico y escriba las ecuaciones ABEL para su diseño.

8.49 Repita el ejercicio anterior utilizando VHDL.

8.50 Escriba un programa VHDL para un contador de n bits que realiza una secuencia de conteo similar a la del ejercicio 8.46. Escriba el programa de manera tal que el tamaño del contador pueda ser cambiado mediante la modificación del valor de una constante simple `N`.

8.51 Modifique el programa VHDL en la tabla 8-14 de manera que el tipo de puertos `D` y `Q` sea `STD_LOGIC_VECTOR`, incluyendo las funciones de conversión que sean requeridas.

8.52 Modifique el programa en la tabla 8-16 para utilizar VHDL estructural, de modo que conforme exactamente el circuito en la figura 8-45, incluyendo los nombres de señal mostrados en la figura. Defina y haga uso de cualquiera de las entidades siguientes que ya no existen en su librería `AND2`, `INV`, `NOR2`, `OR2`, `XNOR2`, `Vdffqqn`.

8.53 Modifique el programa en la tabla 8-17 para utilizar la instrucción generic de VHDL, de manera que el tamaño del contador pueda modificarse utilizando la definición `generic`.

8.54 Diseñe un circuito de conversión paralelo en serie con ocho enlaces serie de 32 canales y 2.048 Mbps, y un solo bus de datos en paralelo de 8 bits y 2.048 MHz que conduzca 256 bytes por cuadro. Cada enlace serie debería tener el formato de cuadro definido en la figura 8-55. Cada línea de datos serie SDATAi debería tener su propia señal de sincronía SYNCi; los pulsos de sincronía deberían estar alternados de modo que SYNCi + 1 tenga un pulso un tic después de SYNCi.

8.55 Muestre la temporización del bus en paralelo y los enlaces en serie, y escriba una tabla o fórmula que muestre cuáles segmentos de tiempo del bus en paralelo son transmitidos en qué enlaces en serie y segmentos de tiempo. Dibuje un diagrama lógico para el circuito utilizando partes MSI de este capítulo; usted puede abreviar elementos repetidos (por ejemplo, registros de corrimiento), mostrando solamente las conexiones únicas para cada una.

8.56 Repita el ejercicio 8.54, suponiendo que todas las líneas de datos en serie deben referenciar sus datos a una sola señal SYNC común. ¿Cuántos chips más requiere este diseño?

8.57 Muestre cómo mejorar el circuito en serie a paralelo del ejercicio 8-57 de modo que el byte recibido en cada segmento de tiempo sea almacenado en su propio registro por 125 μs, hasta que el siguiente byte de ese segmento de tiempo sea recibido. Dibuje el contador y lógica de decodificación para 32 segmentos de tiempo de manera detallada, además de los registros de datos en paralelo y conexiones para los segmentos de tiempo 31, 0 y 1. También dibuje un diagrama de temporización en el estilo de la figura 8-58 que muestra las señales de datos y decodificación asociadas con los segmentos de tiempo 31, 0 y 1.

8.58 Supongamos que a usted se le ha solicitado diseñar una computadora en serie, una que mueve y procesa los datos un bit a la vez. La primera decisión que usted debe tomar es cuál bit transmitir y procesar primero, el LSB o el MSB. ¿Cuál de ellos elegiría, y por qué?

8.59 Diseñe un contador en anillo de autocorrección cuyos estados son 11111110, 11111101, ..., 01111111, utilizando solamente dos encapsulados SSI/MSI.

8.60 Diseñe dos contadores diferentes de 2 bits y 4 estados, donde cada diseño utiliza solamente un encapsulado 74x74 (dos flip-flops [dispositivos biestables] D disparados por flanco) y ninguna otra compuerta.

8.61 Diseñe un contador Johnson de 4 bits y decodificación para los ocho estados utilizando solamente cuatro flip-flops (dispositivos biestables) y ocho compuertas. Su contador no necesita ser de autocorrección.

8.62 Demuestre que un número par de salidas de registro de corrimiento debe conectarse al circuito de paridad impar en un contador LFSR de n bits si genera una secuencia de longitud máxima (Advierta que éste es un requerimiento necesario pero no suficiente. También, aunque la tabla 8-21 es consistente con lo que usted está suponiendo demostrar, ¡citar simplemente la tabla no es una demostración!)

8.63 Demuestre que X0 debe aparecer en el lado derecho de cualquier ecuación de retroalimentación LFSR que genere una secuencia de longitud máxima. (*Nota:* Suponga que la ordenación de bit LFSR y dirección de corrimiento son como las dadas en el texto; es decir, el LFSR se corre a la derecha, hacia la etapa X0).

8.64 Supongamos que un contador LFSR de n bits está diseñado de acuerdo con la figura 8-68 y la tabla 8-21. Demuestre que si el circuito de paridad impar se cambia a un circuito de paridad par, el circuito resultante es un contador que visita $2^n - 1$ etapas, incluyendo todos los estados excepto 11...11.

8.65 Encuentre una ecuación de retroalimentación para un contador LFSR de 3 bits, aparte del dado en la tabla 8-21, que produzca una secuencia de longitud máxima.

8.66 Dado un contador LFSR de n bits que genera una secuencia de longitud máxima ($2^n - 1$ estados), demuestre que una compuerta XOR extra y una compuerta NOR de $n - 1$ entradas conectadas como se sugiere en la figura 8-69 producen un contador con 2^n estados.

8.67 Demuestre que una secuencia de 2^n estados se obtiene todavía si se sustituye una compuerta NAND por una NOR arriba, pero la secuencia de estado es diferente.

8.68 Diseñe un circuito iterativo para verificar la paridad de una palabra de datos de 16 bits con un solo bit de paridad par. ¿Importa el orden de transmisión del bit?

8.69 Modifique el programa del registro de corrimiento en la tabla 8-23 para proporcionar una entrada de borrado asincrónica utilizando un 22V10.

8.70 Escriba un programa ABEL que suministre la misma funcionalidad que un registro de corrimiento 74x299. Demuestre cómo ajustar esta función en un 22V10 simple, o explique por qué no se puede hacer.

8.71 Determine el número de términos de producto requeridos para cada salida del PLD RING8 en la tabla 8-25. ¿Entrará en un 16R8 o en un 16V8R?

8.72 ¿En qué situaciones los programas ABEL de las tablas 8-26 y 8-27 dan diferentes resultados operacionales?

8.73 Modifique el programa ABEL en la tabla 8-26 de manera que las fases sean siempre de por lo menos dos tics de reloj de extensión, incluso si RESTART es asertiva al principio de una fase. RESET debería tener efecto inmediatamente.

8.74 Repita el ejercicio anterior para el programa de la tabla 8-27.

8.75 Suponga que el generador de temporización de la tabla 8-26 se utiliza para controlar un sistema de memoria dinámica, de modo que todas las seis fases deben completarse para leer o escribir la memoria. Si el generador de temporización es reestablecido durante una operación de escritura sin completar la totalidad de las seis fases, el contenido de la memoria puede corromperse. Modifique las ecuaciones en la tabla 8-26 para evitar este problema.

8.76 Un estudiante propuso crear las formas de onda de temporización de la figura 8-72 comenzando con el programa ABEL en la tabla 8-27 y cambiando la codificación de cada uno de los estados P1F, P2F, ..., P6F de manera que la salida de fase correspondiente es 1 en vez de 0, de modo que la salida de fase es 0 solamente durante el segundo tic de cada fase, como se requiere. ¿Es esto un buen enfoque? Haga un comentario sobre los resultados producidos por el compilador ABEL cuando intente esto.

8.77 Las formas de onda de salida producidas por los programas ABEL en las tablas 8-29 y 8-30 no son idénticas, cuando las entradas RESTART y RUN son cambiadas. Explique la razón para esto y posteriormente modifique el programa en la tabla 8-30 de modo que su comportamiento satisfaga el de la tabla 8-29.

8.78 La implementación del contador en anillo ABEL en la tabla 8-26 no es autosincronizante. Por ejemplo, describa qué ocurre si las salidas [P1_L..P6_L] se encuentran inicialmente en 0, y la entrada RUN es asertiva sin activar RESET o RESTART. ¿Qué otros estados de arranque exhiben esta clase de comportamiento no autosincronizante? Modifique el programa de manera que *sea* autosincronizante.

8.79 Repita el ejercicio anterior para la implementación del contador en anillo VHDL en la tabla 8-33.

8.80 Diseñe un circuito iterativo con una entrada B_i por etapa y dos entradas de frontera X y Y de modo que X = 1 si por lo menos dos entradas B_i son 1 y además Y = 1 si por lo menos entradas B_i *consecutivas* son 1.

8.81 Diseñe una máquina de cerradura de combinación de acuerdo a la tabla de estado 7-14 con un contador 74x163 y lógica combinacional para las entradas LD_L, CLR_L y A–D del '163. Utilice valores de contador 0-7 para los estados A–H.

8.82 Escriba un programa ABEL correspondiente al diagrama de estado en la figura 8-84 para la unidad de control del multiplicador.

8.83 Escriba un programa VHDL correspondiente al diagrama de estado en la figura 8-84 para la unidad de control del multiplicador.

8.84 Escriba un programa VHDL que se desempeñe con las mismas entradas, salidas y funciones que la unidad de datos del multiplicador en la figura 8-82.

8.85 Escriba un programa VHDL que combine los programas de los dos anteriores ejercicios para formar un multiplicador completo de corrimiento y suma de 8 bits.

8.86 El texto establece que el diseñador no necesita preocuparse de ningún problema de temporización en el diseño sincrónico de la figura 8-83. En realidad, si existe una pequeña preocupación. Examine las especificaciones de temporización para el 74x377 y discuta al respecto.

8.87 Determine el mínimo periodo de reloj para el circuito multiplicador de corrimiento y suma en la figura 8-83, suponiendo que la máquina de estado está realizada con un solo GAL16V8-20 y que las partes MSI son todas TTL 74LS. Haga uso de la información de temporización del peor de los casos, dada en las tablas en este libro. Para el '194, t_{pd} es, desde el reloj hasta cualquiera de las salidas, de 26 ns y t_s es de 20 ns para las entradas de datos serie y paralelo y de 30 ns para entradas en modo de control.

8.88 Diseñe una unidad de datos y una máquina de estado de unidad de control para multiplicar números de complemento a dos de 8 bits utilizando el algoritmo discutido en la sección 2.8.

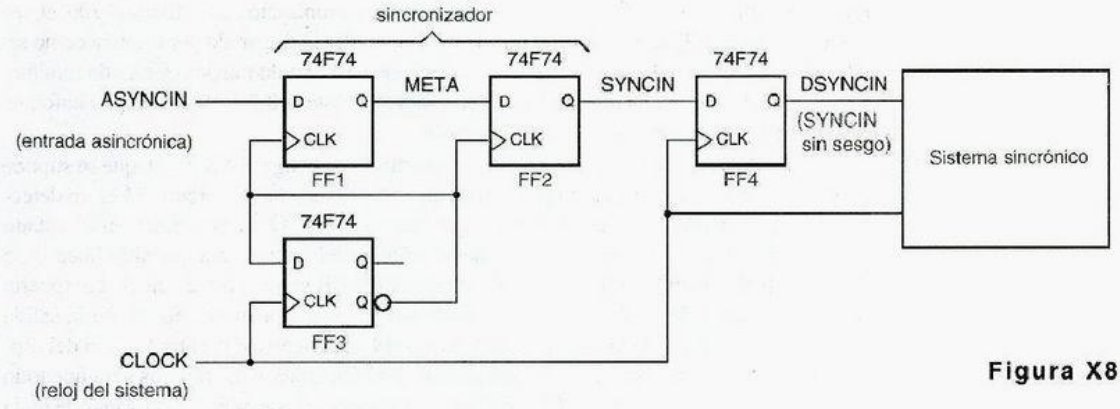


Figura X8.91

8.89 Diseñe una unidad de datos y una máquina de estado de unidad de control para dividir números sin signo de 8 bits aplicando el algoritmo de corrimiento y resta que se analizó en la sección 2.9.

8.90 Suponga que la señal SYNCIN del problema 8.21 se conecta a un circuito combinacional en el sistema síncrónico, el cual a su vez controla las entradas D de los flip-flops (dispositivos biestables) 74ALS74 que están temporizados por CLOCK. ¿Cuál es el máximo retardo de propagación permisible de la lógica combinacional?

8.91 El circuito de la figura X8.91 incluye un flip-flop (dispositivo biestable) eliminador de sesgo de manera que la salida sincronizada del sincronizador de ciclo múltiple se encuentre disponible tan pronto como es posible después del flanco de CLOCK. Ignorando consideraciones de metaestabilidad, ¿cuál es la frecuencia máxima de CLOCK? Suponga que para un 74F74, $t_{\text{establecimiento}} = 5$ ns y $t_{pd} = 7$ ns.

8.92 Aplicando la máxima frecuencia de reloj que se determinó en el ejercicio 8.91, y suponiendo una velocidad de transición asincrónica de 4 MHz, determine el MTBF del sincronizador.

8.93 Determine el MTBF del sincronizador de la figura X8.91, suponiendo una velocidad de transición asincrónica de 4 MHz y una frecuencia de reloj de 40 MHz, que es menor que la máxima determinada en la figura X8.91. En esta situación, se presenta en realidad una "falla del sincronizador" solamente si DSYNCIN es metaestable. En otras palabras, SYNCIN puede permitirse ser metaestable durante un tiempo corto, mientras no afecte DSYNCIN. Esto produce un MTBF mejor.

8.94 Examine la patente de U.S. número 4,999,528, "Metastable-proof flip-flop (dispositivo biestable)" y describa por qué no siempre funciona como se anuncia. (*Sugerencias:* Las patentes pueden hallarse en `www.patents.ibm.com`. Hay suficiente información en este resumen de patente para averiguar cómo pude fallar el circuito.)

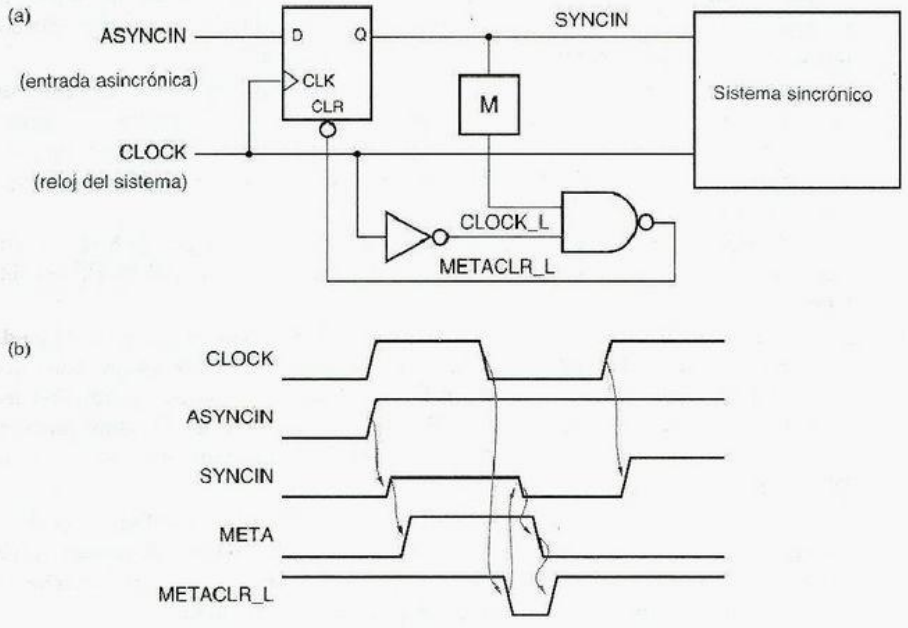
8.95 En el circuito de sincronización de las figuras 8-102, 8-104 y 8-106, se puede reducir el retardo de la transferencia de un byte desde el dominio RCLK hasta el dominio SCLK si se utiliza una versión más anterior del pulso SYNC para arrancar el sincronizador. Suponiendo que usted pueda generar SYNC durante cualquier bit del byte recibido, ¿cuál bit debería utilizar para minimizar el retardo? También determine si su solución satisface los requerimientos de retardo máximo para el circuito. Suponga que todos los componentes tienen temporización 74AHCT que el latch S-R está construido a partir de un par de compuertas NOR acopladas en cruz, y muestre un detallado análisis de temporización para sus respuestas.

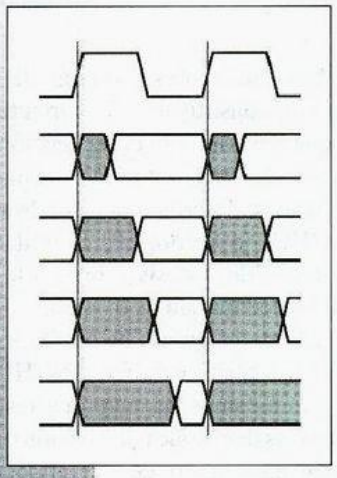
8.96 En vez de utilizar un latch en el circuito de control de sincronización de la figura 8-106, algunas aplicaciones utilizan un flip-flop (dispositivo biestable) D disparado por frontera como se muestra en la figura 8-111. Obtenga los requerimientos de retardo máximo y retardo mínimo para este circuito, correspondiente a las ecuaciones 8-1 hasta la 8-3, y discuta si este enfoque facilita o empeora los requerimientos de retardo.

8.97 Un famoso diseñador digital ideó el circuito mostrado en la figura X8.97(a), que se supone elimina la metaestabilidad en un periodo de un reloj de sistema. El circuito M es un detector de voltaje analógico sin memoria cuya salida es 1 si Q se encuentra en el estado metaestable y 0 de otro modo. La idea del diseñador del circuito era que si la línea Q se detecta que está en el estado metaestable cuando CLOCK se va al nivel bajo, la compuerta NAND borrará el flip-flop (dispositivo biestable) D, el cual a su vez eliminará la salida metaestable, provocando una salida 0 del circuito M y negando así la entrada CLR del flip-flop (dispositivo biestable). Los circuitos son todos suficientemente rápidos para que todo esto ocurra mucho antes de que CLOCK se vaya al nivel alto de nuevo; las formas de onda esperadas se ilustran en la figura X8.97(b).

Desafortunadamente, el sincronizador aún fallará ocasionalmente, y el famoso diseñador digital ahora se encuentra diseñando bolsillos para pantalones de mezclilla. Explique, con detalle, cómo fue que llegó a fallar, incluyendo un diagrama de temporización.

Figura X8.97





capítulo 9

Ejemplos de diseño de circuitos secuenciales

Casi cualquier sistema digital real es un circuito secuencial, todo lo que se necesita es un circuito de retroalimentación, un latch o flip-flop para hacer que el comportamiento actual del circuito dependa de sus entradas pasadas. Sin embargo, si tuviéramos que diseñar o analizar un sistema digital típico, como un circuito secuencial sencillo, terminaríamos con una enorme tabla de estado. Por ejemplo, estrictamente hablando, una PC con solamente 16 Mbytes de memoria principal es un circuito secuencial con ¡aproximadamente $2^{128,000,000}$ estados!

En los capítulos anteriores indicamos que en general analizaremos sistemas digitales en segmentos más pequeños, por ejemplo, dividiéndolos en trayectorias de datos, registros y unidades de control (sección 8.7.1). De hecho, un sistema típico tiene varias unidades funcionales con interfaces bien definidas y conexiones entre ellas (como las soportadas por VHDL, sección 4.7.2) y cada unidad funcional contiene una jerarquía con varias capas de abstracción (como las que respalda tanto VHDL como los paquetes típicos que elaboran diagramas esquemáticos [sección 5.1.6]). De este modo, podemos tratar con circuitos secuenciales en segmentos mucho más pequeños.

Después de toda esta acumulación progresiva, tengo que admitir que el diseño de sistemas digitales jerárquicos complejos se encuentra más allá del ámbito de este texto. Sin embargo, el corazón de cualquier sistema o cualquiera de sus subsistemas es habitualmente una máquina de estado u otro circuito secuencial, y eso es algo que *podemos* estudiar aquí. De esta forma, en este capítulo trataremos de reforzar sus conocimientos acerca del diseño de máquinas de estado y circuitos secuenciales mediante la presentación de diversos ejemplos.

Los primeros diseñadores digitales y muchos diseñadores a lo largo de la década de 1980, escribieron sus tablas de estado a mano y construyeron los circuitos correspondientes utilizando los métodos de síntesis que describimos en la sección 7.4. Sin embargo, difícilmente alguien hace ahora algo parecido. En la actualidad, la mayor parte de las tablas de estado se especifican con un lenguaje de descripción de hardware (HDL, hardware description language) como ABEL, VHDL o Verilog. El compilador HDL efectúa entonces el equivalente de nuestros métodos de síntesis y realiza la máquina especificada en un PLD, CPLD, FPGA, ASIC u otra tecnología de destino.

Este capítulo proporciona ejemplos de diseño de máquinas de estado y otros circuitos secuenciales en dos lenguajes de descripción de hardware diferentes (HDL). En la primera sección ofrecemos ejemplos en ABEL, y enfocamos las máquinas resultantes a PLD pequeños. Algunos de estos ejemplos ilustran las decisiones de división o partición que son necesarias cuando un circuito completo no cabe dentro de un componente simple. En la segunda sección ofrecemos ejemplos en VHDL, que pueden ser aplicados a prácticamente cualquier tecnología.

Como sucedió en el capítulo 6, este capítulo tiene como prerrequisitos solamente los capítulos que lo preceden. Sus dos secciones están escritas para ser bastante independientes entre sí, y el resto del libro está escrito de tal modo que se puede leer este capítulo ahora o saltárselo y regresar a él más tarde.

9.1 Ejemplos de diseño que utilizan ABEL y PLD

En la sección 7.4 diseñamos diversas máquinas de estado simples al traducir sus descripciones en palabras en una tabla de estado, seleccionando una asignación de estado y finalmente sintetizando un circuito correspondiente. Repetimos uno de estos ejemplos que utilizan ABEL y un PLD en la tabla 7-27 y otro en la tabla 7-31. Estos diseños fueron mucho más fáciles de hacer empleando ABEL y un PLD, por dos razones:

- usted no tiene que preocuparse demasiado acerca de la complejidad de las ecuaciones de excitación resultantes, mientras se puedan incluir en el PLD;
- usted puede ser capaz de aprovechar la ventaja de las características del lenguaje ABEL para hacer el diseño más fácil de expresar y de comprender.

Antes de observar más ejemplos, examinemos el comportamiento de temporización y las consideraciones de encapsulado para máquinas de estado que se construyen con los dispositivos PLD.

9.1.1 Temporización y encapsulamiento de máquinas de estado basadas en PLD

La figura 9-1 muestra cómo un PLD genérico que tiene salidas combinacionales y salidas registradas, el cual puede utilizarse en una aplicación de máquina de estado. Los parámetros de temporización t_{CO} y t_{PD} se analizaron en la sección 8.3.3; t_{CO} es el retardo de reloj a salida del flip-flop mientras que t_{PD} es el retardo a través del arreglo AND-OR.

Las variables de estado se asignan a las salidas registradas, naturalmente, y son estables al tiempo t_{CO} después de la frontera ascendente de CLOCK. Las salidas tipo Mealy se asignan a las salidas combinacionales y son estables al tiempo t_{PD} despues de cualquier

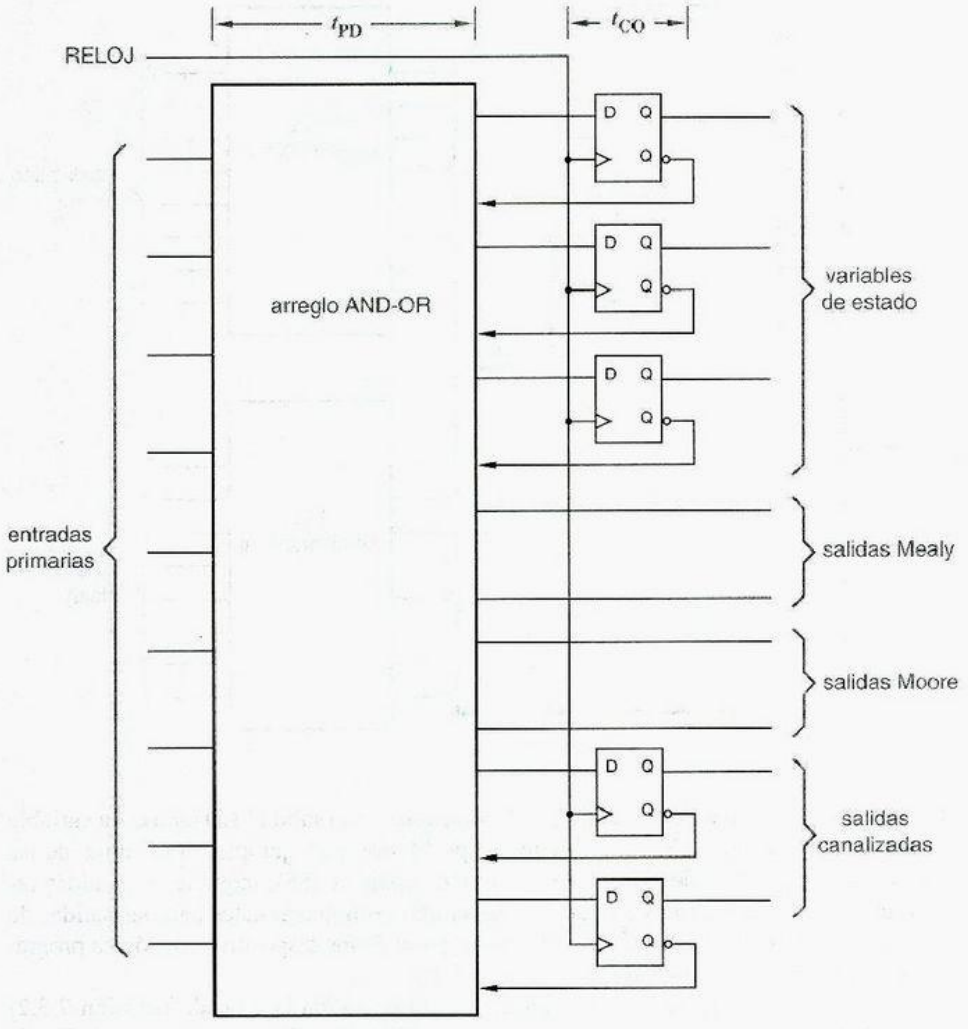


Figura 9-1
Estructura y temporización de un PLD que se utiliza como máquina de estado.

cambio en la entrada que pudiera afectarlas. Las salidas tipo Mealy también pueden modificarse después de un cambio de estado, en cuyo caso llegan a ser estables en el tiempo $t_{CO} + t_{PD}$ después del RELOJ.

Por definición, una salida tipo Moore es una función lógica combinacional del estado actual, por lo que es estable en el tiempo $t_{CO} + t_{PD}$ después de RELOJ. Así, las salidas Moore pueden no ofrecer ninguna ventaja de velocidad sobre las salidas Mealy en el desarrollo de un circuito com PLD. Para obtener un retardo de propagación más rápido, definimos y utilizamos salidas canalizadas en las secciones 7.3.2 y 7.11.5. En una realización PLD, estas salidas vienen directamente de una salida de flip-flop y así tienen un retardo de solamente t_{CO} del reloj. Aparte de tener un retardo más corto, también se garantiza que estén libres de interferencias, algo que es importante en algunas aplicaciones.

Los diseños de máquinas de estado que están basadas en PLD se ven limitados por el número de terminales de entrada y de salida que están disponibles en un PLD sencillo.

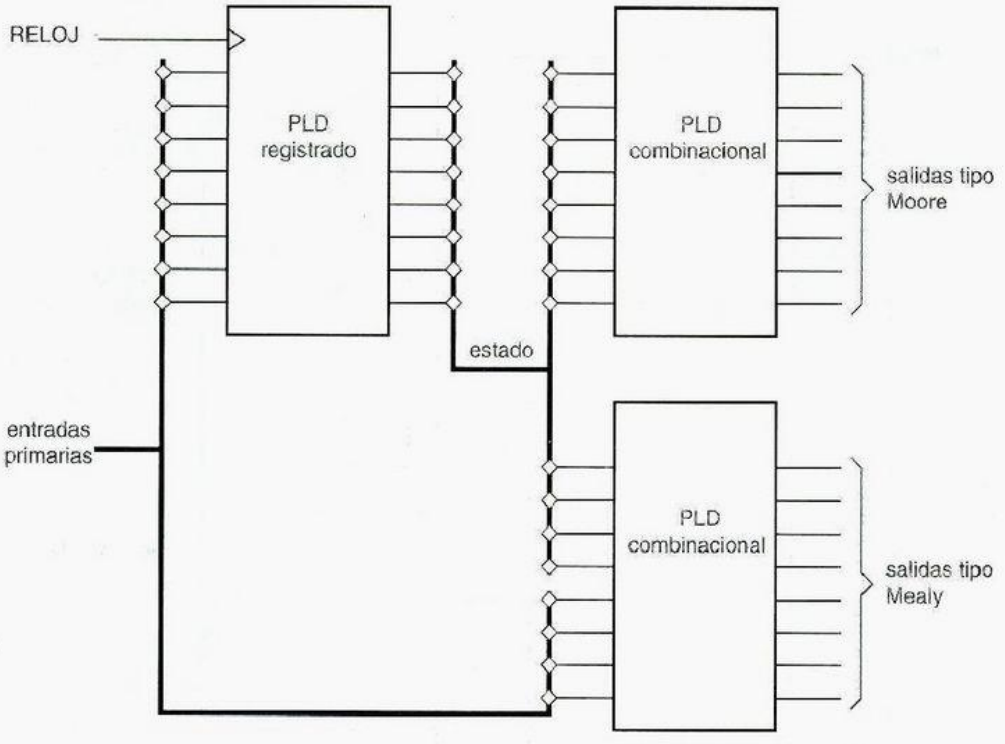


Figura 9-2
Dividiendo un diseño de máquina de estado en tres PLD.

De acuerdo con el modelo de la figura 9-1, se requiere una salida PLD para cada variable de estado y para cada salida tipo Mealy o tipo Moore. Por ejemplo, la máquina de las luces traseras del Thunderbird, de la sección 7.5 (página 585), requiere tres salidas registradas para las variables de estado y seis salidas combinacionales para las salidas de las lámparas. Esto es demasiado para la mayor parte de los dispositivos de lógica programable que hemos descrito, excepto para el 22V10.

Por el otro lado, una asignación de estado de salida codificada (sección 7.3.2) requiere de una menor cantidad total de salidas PLD. Si utilizamos una asignación de estado de salida codificada, la máquina de luces traseras del Thunderbird puede construirse en un solo 16V8, como mostraremos en la sección 9.1.3.

Como sucede con cualquier variable de estado o salida tipo Moore, una variable de estado de salida codificada es estable al tiempo t_{CO} después del flanco del reloj. De este modo, una asignación de estado de salida codificada mejora la velocidad (además de la eficiencia) del encapsulado. En las luces traseras del Thunderbird, el hecho de activar las luces intermitentes de emergencia con una anticipación de 10 ns no tiene mayor importancia, pero en un sistema digital de alto rendimiento, un retardo extra de 10 ns podría hacer la diferencia entre una velocidad máxima de reloj del sistema de 100 MHz y un máximo de solamente 50 MHz.

Si un diseño *debe* dividirse en dos o más PLD, la mejor división en términos tanto de simplicidad de diseño como de eficiencia de encapsulado es con frecuencia la que se muestra en la figura 9-2. Un PLD secuencial simple se utiliza para crear el comportamiento de estado siguiente requerido, y los PLD combinacionales se emplean para crear tanto salidas tipo Mealy como tipo Moore de las variables de estado y entradas.

ALIVIO PARA UN DOLOR DE CABEZA

Las modernas herramientas para el diseño de sistemas digitales basado en PLD pueden eliminar algo del ensayo y error que de otra manera puede asociarse con el ajuste de un diseño dentro de una conjunto de PLD. Para utilizar esta capacidad, el diseñador introduce las ecuaciones y descripciones de máquina de estado para el diseño, sin especificar necesariamente los dispositivos y terminales externas reales que deberían emplearse para realizar el diseño. Una herramienta de software denominada *particionador* intenta dividir el diseño en el número más pequeño posible de dispositivos de una familia determinada, mientras que minimiza el número de terminales que se utilizan para la comunicación entre los dispositivos. La partición o división puede ser completamente automática parcial o totalmente controlada por el mismo usuario.

Con cierta frecuencia, los dispositivos más grandes (CPLD y FPGA) presentan limitantes internas en su arquitectura que pueden crear dolores de cabeza cuando no está presente el experto en software. El diseñador puede tener la impresión, basado en entrada, salida y los requerimientos de lógica combinacional total y de flip-flops, que un diseño se ajustará en un solo CPLD o FPGA. Sin embargo, el diseño aún debe dividirse entre múltiples PLD o bloques lógicos dentro del dispositivo más grande, donde cada bloque tiene solamente una funcionalidad limitada.

Por ejemplo, una salida que requiere muchos términos de producto puede tener que robarse algunas de las salidas físicamente adyacentes. Esto puede a su vez afectar si las terminales adyacentes se pueden utilizar como entradas o salidas, y cuántos términos de producto se encuentren disponibles para ellos. También puede afectar la habilidad de interconectar señales entre los bloques cercanos y el retardo en el peor de los casos de estas señales. Todas estas limitantes se pueden rastrear si se utiliza un software *de ajuste* que aplica un enfoque heurístico para hallar la mejor división de las funciones entre los bloques dentro de un solo CPLD o FPGA.

En muchos entornos de diseño, el software particionador o ajustador trabajan en conjunto y de manera interactiva con el diseñador para encontrar una realización aceptable empleando un conjunto de PLD, CPLD y FPGA.

9.1.2 Varias máquinas simples

En la sección 7.4 diseñamos varias máquinas de estado simple aplicando los métodos tradicionales. Primero presentamos un diseño basado en PLD y ABEL para el primero de éstos, en la figura 7-25 y posteriormente, mostramos cómo podríamos hacer más comprensible el funcionamiento de la máquina al hacer mejor uso de las características de ABEL en la tabla 7-27.

Nuestro segundo ejemplo fue una "máquina de conteo de unos" con la siguiente especificación:

> Diseñe una máquina de estado sincrónica temporizada con dos entradas, X y Y, y una salida, Z. La salida debería ser 1 si el número de entradas 1 en X y Y desde el restablecimiento es un múltiplo de 4, y 0 de otro modo.

Desarrollamos una tabla de estado para esta máquina en la tabla 7-12. Sin embargo, podemos expresar la función de la máquina mucho más fácilmente en ABEL, como lo mostramos en la tabla 9-1. Note que para esta máquina obtuvimos mejores resultados cuando no utilizamos la sintaxis de "diagrama de estado" de ABEL. Podríamos expresar la función de la máquina de manera más natural utilizando una instrucción anidada

Tabla 9-1
Programa ABEL para la máquina de estado de conteo de unos.

```
module onesctsm
title 'Ones-counting State Machine'
ONESCTSM device 'P16V8R';

" Inputs and outputs
CLOCK, RESET, X, Y      pin 1, 2, 3, 4;
Z                       pin 13 istype 'com';
COUNT1..COUNT0          pin 14, 15 istype 'reg';

" Sets
COUNT = [COUNT1..COUNT0];

equations
COUNT.CLK = CLOCK;
WHEN RESET THEN COUNT := 0;
ELSE WHEN X & Y THEN COUNT := COUNT + 2;
ELSE WHEN X # Y THEN COUNT := COUNT + 1;
ELSE COUNT := COUNT;

Z = (COUNT == 0);

end onesctsm
```

WHEN y la operación de suma integrada. La primera cláusula WHEN obliga a la máquina a ir a su estado inicial y al conteo de 0 sobre el restablecimiento, y las cláusulas sucesivas incrementan el conteo en 2, 1 o 0 conforme se requiera cuando la máquina no se reestablece. Note que ABEL "saca" el bit de acarreo en la suma, lo cual equivale a efectuar la suma módulo 4. La máquina cabe fácilmente en un dispositivo GAL16V8. Tiene cuatro estados, porque hay dos flip-flops en su realización.

Otro ejemplo de la sección 7.4 es una máquina de estado con cerradura de combinación (a continuación omitimos la salida HINT en la especificación original):

> Diseñe una máquina de estado síncrona temporizada con una entrada, X, y una salida, UNLK. La salida UNLK debería ser 1 si y sólo si X es 0 y la secuencia de entradas recibidas en X en los siete tics de reloj precedentes fue 0110111.

DESACTIVANDO MALOS HÁBITOS

En el capítulo 7 iniciamos el mal hábito de diseñar tablas de estado sin incluir una entrada de restablecimiento explícita. Había una razón para esto: cada entrada adicional duplica potencialmente la cantidad de trabajo que tendríamos que hacer para sintetizar la máquina de estado utilizando métodos manuales, y había poco que ganar pedagógicamente hablando.

Ahora que realizamos el diseño de máquina de estado aplicando lenguajes que utilizan herramientas automatizadas, deberíamos adquirir el hábito de proporcionar siempre una entrada de restablecimiento explícita que envíe a la máquina de vuelta a un estado conocido. ¡Es un requerimiento importante en los diseños reales!

Desarrollamos una tabla de estado para esta máquina en la tabla 7-14, y escribimos un programa ABEL equivalente en la tabla 7-31. Sin embargo, una vez más, podemos tomar un enfoque diferente que sea más fácil de comprender. Para este ejemplo, observamos que la salida de la máquina en cualquier momento se encuentra completamente determinada por sus entradas sobre los ocho tics de reloj precedentes. De este modo, podemos emplear un enfoque denominado "de memoria finita" para diseñar esta máquina, donde de manera explícita rastreamos las siete entradas anteriores y entonces formamos la salida como una función combinacional de estas entradas.

El programa ABEL en la tabla 9-2 aplica el enfoque de la memoria finita. Se escribió utilizando conjuntos para hacer las modificaciones de manera fácil, por ejemplo, cambiar la combinación. Sin embargo, advierta que la salida HINT debería ser tan difícil de proporcionar en esta versión de la máquina como en la original (vea el ejercicio 9.2).

Tabla 9-2 Programa de memoria finita para máquina de estado con cerradura de combinación.

```
module comblckf
title 'Combination-Lock State Machine'
"COMBLCKF device 'P16V8R';

" Input and output pins
CLOCK, RESET, X              pin 1, 2, 3;
X1..X7                       pin 12..18 istype 'reg';
UNLK                         pin 19;

" Sets
XHISTORY = [X7..X1];
SHIFTX   = [X6..X1, X];

equations

XHISTORY.CLK = CLOCK;
XHISTORY := !RESET & SHIFTX;

UNLK = !RESET & (X == 0) & (XHISTORY == [0,1,1,0,1,1,1]);

END comblckf
```

DISEÑO DE MEMORIA FINITA

El enfoque de diseño de memoria finita para el diseño de máquina de estado puede ser generalizado. Las salidas de una máquina están determinadas totalmente por sus entradas y salidas actuales durante los anteriores n tics de reloj, donde n es un entero acotado, finito. Cualquier máquina que pueda desarrollarse como se indica en la figura 9-3 será una máquina de memoria finita. Cabe indicar que una máquina de *estado* finito no necesita ser una máquina de *memoria* finita. Por ejemplo, la máquina de conteo de unos en la tabla 9-1 tiene solamente cuatro estados pero no es una máquina de memoria finita, su salida depende de cada valor de X y Y desde el restablecimiento (reset).

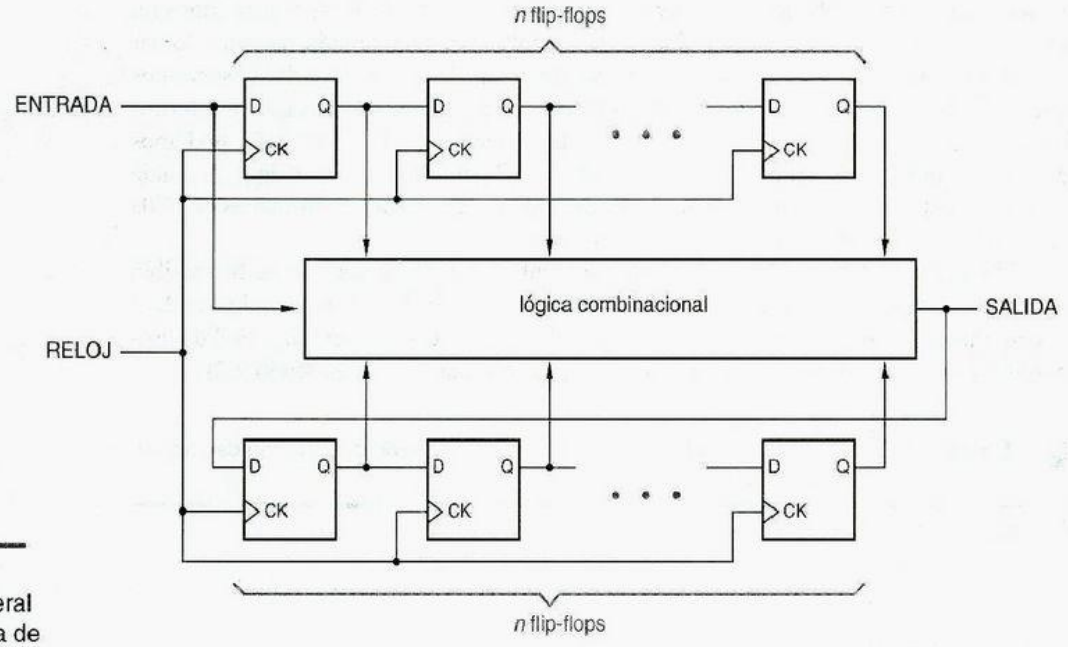


Figura 9-3
Estructura general de una máquina de memoria finita.

9.1.3 Las luces traseras del Ford Thunderbird

Describimos y diseñamos una máquina de estado para las "luces traseras del Thunderbird" de la sección 7.5. La tabla 9-3 es un "diagrama de estado" ABEL equivalente para la máquina de luces traseras del Ford Thunderbird. Existe una estrecha correspondencia entre este programa y el diagrama de estado de la figura 7-64, que utiliza la asignación de estado de la tabla 7-16. Excepto por la entrada RESET (que se agregó), el programa produce exactamente las mismas ecuaciones reducidas que las ecuaciones de transición explícitas resultantes de la lista de transición, la cual resolvimos a mano en la sección 7.6.

El programa de la tabla 9-3 maneja solamente las variables de estado de la máquina de las luces traseras. La lógica de salida requiere seis salidas combinacionales, pero solamente cinco salidas más se encuentran disponibles en el 16V8 que especifica el programa. Se puede utilizar otro PLD para decodificar los estados, aplicando la clase de partición que mostramos en la figura 9-2. De manera alternativa, un PLD más grande, tal como el 22V10, podría proporcionar suficientes salidas para un diseño sencillo con PLD.

Un enfoque aún mejor es reconocer que los valores de salida de la máquina de las luces traseras son diferentes en cada caso, de modo que también podamos emplear una asignación de estado de salida codificada. Esto requiere solamente de seis salidas registradas y no salidas combinacionales de un 16V8, como se ilustra en la figura 9-4. Solamente se deben modificar las definiciones del dispositivo, terminal y estado en el programa ABEL anterior, como se muestra en la tabla 9-4. Cada una de las seis ecuaciones de excitación que obtuvimos utilizan cuatro términos de producto.

Tabla 9-3
Programa ABEL para la máquina de las luces traseras del Ford Thunderbird.

```
module tbirdsd
title 'State Machine for T-Bird Tail Lights'
TBIRDSD device 'P16V8R';

" Input and output pins
CLOCK, LEFT, RIGHT, HAZ, RESET   pin 1, 2, 3, 4, 5;
Q0, Q1, Q2                       pin 14, 15, 16 istype 'reg';

" Definitions
QSTATE = [Q2,Q1,Q0];             " State variables
IDLE   = [ 0, 0, 0];             " States
L1     = [ 0, 0, 1];
L2     = [ 0, 1, 1];
L3     = [ 0, 1, 0];
R1     = [ 1, 0, 1];
R2     = [ 1, 1, 1];
R3     = [ 1, 1, 0];
LR3    = [ 1, 0, 0];

equations
QSTATE.CLK = CLOCK;

state_diagram QSTATE
state IDLE: IF RESET THEN IDLE
            ELSE IF (HAZ # LEFT & RIGHT) THEN LR3
            ELSE IF LEFT THEN L1 ELSE IF RIGHT THEN R1
            ELSE IDLE;
state L1:   IF RESET THEN IDLE ELSE IF HAZ THEN LR3 ELSE L2;
state L2:   IF RESET THEN IDLE ELSE IF HAZ THEN LR3 ELSE L3;
state L3:   GOTO IDLE;
state R1:   IF RESET THEN IDLE ELSE IF HAZ THEN LR3 ELSE R2;
state R2:   IF RESET THEN IDLE ELSE IF HAZ THEN LR3 ELSE R3;
state R3:   GOTO IDLE;
state LR3:  GOTO IDLE;

end tbirdsd
```

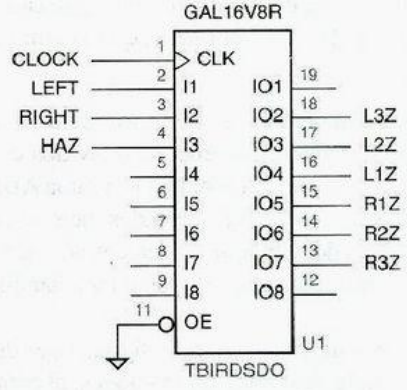


Figura 9-4
Un diseño de PLD simple para las luces traseras del Ford Thunderbird.

Tabla 9-4
Asignación de estado de salida codificada para la máquina de luces traseras del Thunderbird.

```
module tbirdsdo
title 'Output-Coded T-Bird Tail Lights State Machine'
TBIRDSDO device 'P16V8R';

" Input and output pins
CLOCK, LEFT, RIGHT, HAZ, RESET     pin 1, 2, 3, 4, 5;
L3Z, L2Z, L1Z, R1Z, R2Z, R3Z       pin 18..13 istype 'reg';

" Definitions
QSTATE = [L3Z,L2Z,L1Z,R1Z,R2Z,R3Z];  " State variables
IDLE   = [  0,  0,  0,  0,  0,  0];  " States
L3     = [  1,  1,  1,  0,  0,  0];
L2     = [  0,  1,  1,  0,  0,  0];
L1     = [  0,  0,  1,  0,  0,  0];
R1     = [  0,  0,  0,  1,  0,  0];
R2     = [  0,  0,  0,  1,  1,  0];
R3     = [  0,  0,  0,  1,  1,  1];
LR3    = [  1,  1,  1,  1,  1,  1];
```

9.1.4 El juego de las adivinanzas

En la sección 7.7.1 se definió una máquina para "juego de adivinanzas", con la siguiente descripción:

> Diseñe una máquina de estado sincrónica temporizada con cuatro entradas, G1-G4, donde se conectan unos botones. La máquina tiene cuatro salidas, L1-L4, a las que se conectan lámparas o LEDs adicionales que están ubicados cerca de los botones numerados. También hay una salida ERR que se conecta a una lámpara roja. En el funcionamiento normal, las salidas L1-L4 exhiben un patrón 1 de 4. En cada tic de reloj, el patrón se adelanta una posición; la frecuencia de reloj es de aproximadamente 4 Hz.
>
> Las adivinanzas se efectúan presionando un botón, lo que produce una entrada Gi asertiva. Cuando una entrada Gi es asertiva, la salida ERR será asertiva si se presiona el botón "equivocado", es decir, si la entrada Gi que se detecta en el tic del reloj no tiene el mismo número de la salida de la lámpara que se activó antes del tic de reloj. Una vez que se hace una adivinanza, el juego se detiene y la salida ERR mantiene el mismo valor para uno o más tics de reloj hasta que la entrada Gi es negada, y luego el juego se reanuda.

Como discutimos en la sección 7.7.1, la máquina requiere seis estados: cuatro para cada uno de los cuales la lámpara correspondiente se encuentra encendida, y dos cuando el juego se detiene después de presionar con o sin éxito un botón. Un programa ABEL para el juego de las adivinanzas se muestra en la tabla 9-5. Se realizaron dos mejoras para perfeccionar la capacidad de prueba y la operación de la máquina: una entrada RESET que obliga al juego a que tome un estado de inicio conocido y dos estados sin utilizar que tienen transiciones explícitas al estado de inicio.

La máquina del juego de las adivinanzas utiliza las mismas asignaciones de estado que la versión original en la sección 7.7.1. Empleando estas asignaciones, el compilador

Tabla 9-5
Programa ABEL para la máquina del juego de adivinanzas.

```
module ggame
Title 'Guessing-Game State Machine'
GGAME device 'P16V8R';

" Inputs and outputs
CLOCK, RESET, G1..G4          pin 1, 2, 3..6;
L1..L4, ERR                   pin 12..15, 19 istype 'com';
Q2..Q0                        pin 16..18 istype 'reg';

" Sets
G = [G1..G4];
L = [L1..L4];

" States
QSTATE = [Q2,Q1,Q0];
S1     = [ 0, 0, 0];
S2     = [ 0, 0, 1];
S3     = [ 0, 1, 1];
S4     = [ 0, 1, 0];
SOK    = [ 1, 0, 0];
SERR   = [ 1, 0, 1];
EXTRA1 = [ 1, 1, 0];
EXTRA2 = [ 1, 1, 1];

state_diagram QSTATE

state S1:  IF RESET THEN SOK ELSE IF G2 # G3 # G4 THEN SERR
           ELSE IF G1 THEN SOK ELSE S2;

state S2:  IF RESET THEN SOK ELSE IF G1 # G3 # G4 THEN SERR
           ELSE IF G2 THEN SOK ELSE S3;

state S3:  IF RESET THEN SOK ELSE IF G1 # G2 # G4 THEN SERR
           ELSE IF G3 THEN SOK ELSE S4;

state S4:  IF RESET THEN SOK ELSE IF G1 # G2 # G3 THEN SERR
           ELSE IF G4 THEN SOK ELSE S1;

state SOK:  IF RESET THEN SOK
            ELSE IF G1 # G2 # G3 # G4 THEN SOK ELSE S1;

state SERR:  IF RESET THEN SOK
             ELSE IF G1 # G2 # G3 # G4 THEN SERR ELSE S1;

state EXTRA1:  GOTO SOK;
state EXTRA2:  GOTO SOK;

equations

QSTATE.CLK = CLOCK;

L1  = (QSTATE == S1);
L2  = (QSTATE == S2);
L3  = (QSTATE == S3);
L4  = (QSTATE == S4);
ERR = (QSTATE == SERR);

end ggame
```

Tabla 9-6
Uso de términos de producto en el PLD de una máquina de estado para el juego de adivinanzas.

```
P-Terms    Fan-in  Fan-out  Type  Name
---------  ------  -------  ----  --------
  1/3        3        1     Pin   L1
  1/3        3        1     Pin   L2
  1/3        3        1     Pin   L3
  1/3        3        1     Pin   L4
  1/3        3        1     Pin   ERR
  6/2        7        1     Pin   Q2.REG
  1/7        7        1     Pin   Q1.REG
 11/8        8        1     Pin   Q0.REG
=========
 23/32          Best P-Term Total: 16
                      Total Pins: 14
              Average P-Term/Output: 2
```

ABEL arranca ecuaciones minimizadas con el número de términos de producto que se indican en la tabla 9-6. La salida Q0 apenas cabe en un GAL16V8 (ocho términos de producto). Si necesitamos ahorrar términos, la forma en la que hemos escrito el programa nos permite intentar alternar asignaciones de estado (vea el ejercicio 9.4).

Una alternativa más productiva puede ser intentar una asignación de estado de salida codificada. Podemos emplear un bit de estado/salida por lámpara (L1 . . L4) y utilizar un bit más (ERR) para distinguir entre los estados SOK y SERR cuando las lámparas están compoletamente apagadas. Esto nos permite reducir las ecuaciones para L1 . . L4 y ERR de la tabla 9-5. La nueva asignación se muestra en la tabla 9-7. Con esta asignación, L1 utiliza dos términos de producto y L2 . . L4 emplean solamente un término de producto cada una. Desgraciadamente, la salida ERR se expande en 16 términos de producto.

Tabla 9-7
Definiciones ABEL para la máquina del juego de las adivinanzas con una asignación de estado de salida codificada.

```
module ggameoc
Title 'Guessing-Game State Machine'
GGAMEOC device 'P16V8R';

" Inputs and outputs
CLOCK, RESET, G1..G4          pin 1, 2, 3..6;
L1..L4, ERR                   pin 12..15, 18 istype 'reg';

" States
QSTATE = [L1,L2,L3,L4,ERR];
S1     = [ 1, 0, 0, 0,  0];
S2     = [ 0, 1, 0, 0,  0];
S3     = [ 0, 0, 1, 0,  0];
S4     = [ 0, 0, 0, 1,  0];
SOK    = [ 0, 0, 0, 0,  0];
SERR   = [ 0, 0, 0, 0,  1];

...
```

Parte de nuestro problema con esta particular asignación de salida codificada es que no aprovechamos completamente sus propiedades. Note que básicamente se trata de una codificación "activa-uno", pero las definiciones de estado en la tabla 9-7 requieren que se decodifiquen todos los cinco bits de estado para cada estado. Una versión alterna de la codificación que utiliza elementos "sin importancia" se ilustra en la tabla 9-8.

En la nueva versión, suponemos que los bits de estado nunca toman alguna combinación de valores diferente a la que se definió originalmente en la tabla 9-7. Por ejemplo, si vemos que el bit de estado L1 es 1, la máquina debe encontrarse en el estado S1 sin tener en cuenta los valores de cualquier otro bit de estado. Por lo tanto, podemos establecer estos bits como "sin importancia" en la definición de S1 en la tabla 9-8. ABEL establecerá cada X a 0 cuando codifique un estado siguiente, pero tratará cada X como "sin importancia" cuando decodifique el estado actual. De esta forma, debemos *extremar* nuestras precauciones para asegurarnos que los estados codificados sean mutuamente exclusivos, es decir, que ningún estado siguiente y legítimo coincida con dos o más definiciones de estado diferentes. De lo contrario, los resultados compilados no tendrán el comportamiento esperado.

Las ecuaciones reducidas que produce la codificación de salida de la tabla 9-8 utilizan tres términos de producto para L1, cada uno para L2 . . L4, y sólo siete para ERR. De modo que el cambio valió la pena. Sin embargo, debemos recordar que la nueva

```
X = .X.;
QSTATE = [L1,L2,L3,L4,ERR];
S1     = [ 1, X, X, X,  X];
S2     = [ X, 1, X, X,  X];
S3     = [ X, X, 1, X,  X];
S4     = [ X, X, X, 1,  X];
SOK    = [ 0, 0, 0, 0,  0];
SERR   = [ X, X, X, X,  1];
```

Tabla 9-8
Codificación de salida para la máquina del juego de adivinanzas que utiliza elementos "sin importancia".

CÓMO FUNCIONAN LOS ELEMENTOS SIN IMPORTANCIA

Para comprender cómo trabajan los elementos sin importancia en una codificación de estado, primero tenemos que comprender la forma en la que ABEL crea ecuaciones internamente a partir de los diagramas de estado. Dentro de un estado determinado S, cada instrucción de transición (IF-THEN-ELSE o GOTO) hace que los conjuntos de iniciación de ciertas variables de estado se incrementen de acuerdo con la condición de transición. La condición de transición es una expresión que debe ser verdadera para "ir hacia (go to)" ese objetivo, incluyendo la que está en el estado S. Por ejemplo, todas las condiciones especificadas en un estado como S1 en la tabla 9-5 están implícitamente operadas con AND con la expresión "QSTATE==S1". Debido a la manera en que se define S1 utilizando los elementos sin importancia, esta verificación de igualdad genera solamente una sola literal (L1) en lugar de un término AND, conduciendo a una simplificación adicional posterior.

Para cada estado destino en una instrucción de transición, los conjuntos de iniciación de (únicamente) las variables de estado que son 1 en ese estado son aumentadas de acuerdo con la condición de transición. De esta forma, cuando un estado codificado como S1 en la tabla 9-8 aparece como un objetivo en alguna instrucción de transición, solamente el conjunto de iniciación de L1 se aumenta. Esto explica por qué la codificación real del estado S1 como un destino es 100000.

máquina es diferente a la que se muestra en la tabla 9-7. Consideremos ahora lo que sucede si la máquina entra a un estado que no está especificado. En la máquina original (cuya codificación de salida está completamente especificada) no hay estados siguientes para los $2^5 - 6 = 26$ estados que no están especificados, de manera que la máquina de estado siempre irá al estado codificado 00000 (SOK) cuando parte de estados no especificados. En la nueva máquina, los estados "no especificados" en realidad no están sin especificar; por ejemplo, el estado codificado 11111 satisface cinco estados codificados, S1–S4 y SERR. El estado siguiente será realmente el "OR" de los siguientes estados para los estados codificados coincidentes. (Lea el cuadro anterior para entender por qué se presentan estos resultados.) De nueva cuenta, se necesita ser muy cuidadoso.

EXPECTATIVAS DE RESTABLECIMIENTO

Al analizar el programa del juego de adivinanzas en la tabla 9-5, se podría esperar que la entrada RESET llevaría a la máquina al estado SOK, y así ocurre. Sin embargo, el momento en que se tienen estados codificados parcialmente o cuando están sin especificar como en las tablas 9-7 o 9-8, no se debe asumir nada.

Con relación al cuadro de la página anterior, recuerde que las instrucciones de transición de las máquinas de estado ABEL aumentan los conjuntos de iniciación de las variables de estado. Si una combinación de estado que no se utiliza (en particular) no coincide con ninguno de los estados para los que se escribieron las instrucciones de transición, entonces no se aumentará ningún conjunto de iniciación. Así, la única transición desde ese estado será al estado con la codificación con todos ceros.

Por esta razón, es útil codificar el estado de restablecimiento o un estado "seguro" como de ceros en su totalidad. Si esto no es posible, pero el estado de todos ceros se encuentra todavía sin utilizar, se puede proporcionar en forma explícita una transición desde el estado de todos ceros a un estado seguro deseado.

9.1.5 Reinvención de los controladores de semáforos

Nuestro ejemplo final proviene del mundo de los automóviles y el tráfico vehicular. En California, especialmente en la bella ciudad de Sunnyvale, los controladores de los semáforos están cuidadosamente diseñados para *maximizar* el tiempo de espera de los automóviles en las intersecciones o cruces. Uno de tantos cruces que se utilza con poca frecuencia (el cual tendría no más de un signo "ceder el paso" si estuviera en Chicago) tendrá los sensores y señales que se muestran en la figura 9-5. La máquina de estado que controla las señales de tráfico utiliza un reloj de 1 Hz y un temporizador, además tiene cuatro entradas:

NSCAR Asertiva cuando un automóvil que circula de norte a sur se encuentra sobre cualquier sensor en cualquier lado del cruce.

EWCAR Asertiva cuando un automóvil que circula de este a oeste se encuentra sobre cualquier sensor en cualquier lado del cruce.

TMLONG Asertiva si han pasado más de cinco minutos desde que el temporizador arrancó; permanece así hasta que se restablece el temporizador.

TMSHORT Asertiva si han pasado más de cinco segundos desde que el temporizador arrancó; permanece así hasta que se reestablece el temporizador.

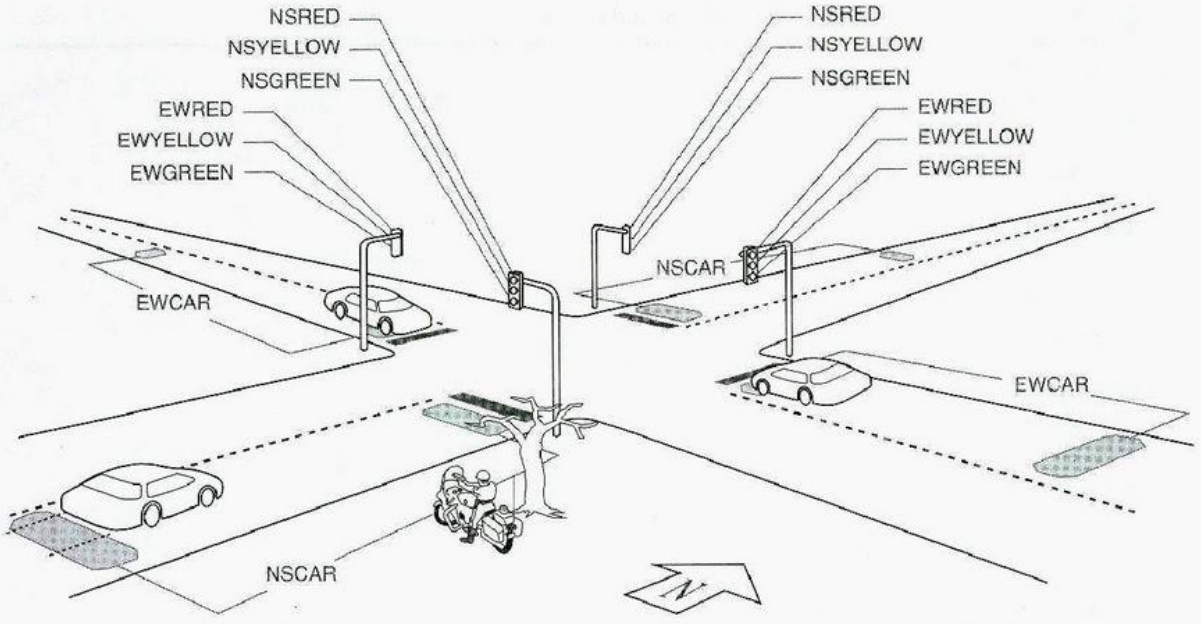


Figura 9-5 Señales y sensores de tráfico en un cruce de caminos en Sunnyvale, California.

La máquina de estado tiene siete salidas:

NSRED, NSYELLOW y NSGREEN Controlan las luces de norte a sur.

EWRED, EWYELLOW y EWGREEN Controlan las luces de este a oeste.

TMRESET Cuando se afirma, reestablece el temporizador y niega TMSHORT y TMLONG. El temporizador inicia el cronometraje cuando TMRESET es negada.

Un algoritmo típico, que cuenta con la aprobación de las autoridades municipales para controlar los semáforos, está integrado en el programa ABEL de la tabla 9-9. Este algoritmo produce dos comportamientos que se ven con frecuencia en los semáforos "inteligentes". En la noche, cuando el tráfico es ligero, mantiene un automóvil detenido en el semáforo hasta por cinco minutos, a menos que un automóvil se aproxime en la calle transversal, en cuyo caso detiene el tráfico del cruce y deja que el automóvil que estaba esperando avance. (El sensor de "advertencia anticipada" se encuentra lo suficientemente atrás para cambiar las luces del semáforo antes que el automóvil que se aproxima alcance el cruce.) Durante el día, cuando el tráfico es pesado y siempre se tienen automóviles esperando en ambas direcciones, se intercalan las señales luminosas cada cinco segundos, minimizando de este modo el uso del cruce y maximizando el tiempo de espera de todos, creando por ello un clamor generalizado para más impuestos para arreglar el problema.

Las ecuaciones para la salida TMRESET son dignas de notarse. Esta salida es asertiva durante los estados de "doble rojo", NSDELAY y EWDELAY, para restablecer el temporizador y prepararlo para el siguiente ciclo verde. La señal de salida deseada podría generarse en una terminal de salida combinacional al decodificar estos dos estados, pero en vez de esto hemos optado por generarla en una terminal de salida registrada al decodificar los *predecesores* de estos dos estados.

Tabla 9-9 Programa de los semáforos de Sunnyvale.

```
module svaletl
title 'State Machine for Sunnyvale, CA, Traffic Lights'
SVALETL device 'P16V8R';

" Input and output pins
CLOCK, !OE                          pin 1, 11;
NSCAR, EWCAR, TMSHORT, TMLONG       pin 2, 3, 8, 9;
Q0, Q1, Q2, TMRESET_L               pin 17, 16, 15, 14 istype 'reg';

" Definitions
LSTATE  = [Q2,Q1,Q0];               " State variables
NSGO    = [ 0, 0, 0];               " States
NSWAIT  = [ 0, 0, 1];
NSWAIT2 = [ 0, 1, 1];
NSDELAY = [ 0, 1, 0];
EWGO    = [ 1, 1, 0];
EWWAIT  = [ 1, 1, 1];
EWWAIT2 = [ 1, 0, 1];
EWDELAY = [ 1, 0, 0];

state_diagram LSTATE
state NSGO:                         " North-south green
   IF (!TMSHORT) THEN NSGO          " Minimum green is 5 seconds.
   ELSE IF (TMLONG) THEN NSWAIT     " Maximum green is 5 minutes.
   ELSE IF (EWCAR & !NSCAR)         " If E-W car is waiting and no one
         THEN NSGO                  "  is coming N-S, make E-W wait!
   ELSE IF (EWCAR & NSCAR)          " Cars coming in both directions?
         THEN NSWAIT                " Thrash!
   ELSE IF (!NSCAR)                 " Nobody coming N-S and not timed out?
         THEN NSGO                  " Keep N-S green.
   ELSE NSWAIT;                     " Else let E-W have it.

state NSWAIT: GOTO NSWAIT2;         " Yellow light is on for two ticks for safety.
state NSWAIT2: GOTO NSDELAY;        " (Drivers go 70 mph to catch this turkey green!)
state NSDELAY: GOTO EWGO;           " Red in both directions for added safety.

state EWGO: " East-west green; states defined analogous to N-S
   IF (!TMSHORT) THEN EWGO
   ELSE IF (TMLONG) THEN EWWAIT
   ELSE IF (NSCAR & !EWCAR) THEN EWGO
   ELSE IF (NSCAR & EWCAR) THEN EWWAIT
   ELSE IF (!EWCAR) THEN EWGO ELSE EWWAIT;

state EWWAIT: GOTO EWWAIT2;
state EWWAIT2: GOTO EWDELAY;
state EWDELAY: GOTO NSGO;

equations

LSTATE.CLK = CLOCK;  TMRESET_L.CLK = CLOCK;
!TMRESET_L := (LSTATE == NSWAIT2)   " Reset the timer when going into
            + (LSTATE == EWWAIT2); "   state NSDELAY or state EWDELAY.
end svaletl
```

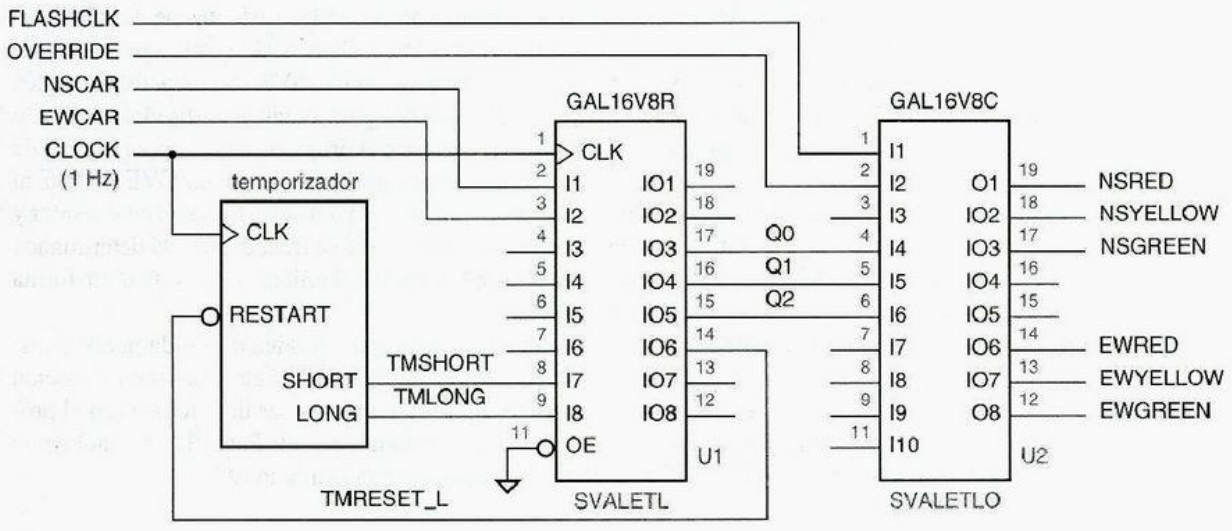


Figura 9-6 Controlador de semáforo de Sunnyvale que utiliza dos PLD.

Tabla 9-10 Lógica de salida para los semáforos de Sunnyvale.

```
module svaletlo
title 'Output logic for Sunnyvale, CA, Traffic Lights'
SVALETLO device 'P16V8C';

" Input pins
FLASHCLK, OVERRIDE, Q0, Q1, Q2   pin 1, 2, 4, 5, 6;

" Output pins
NSRED, NSYELLOW, NSGREEN         pin 19, 18, 17 istype 'com';
EWRED, EWYELLOW, EWGREEN         pin 14, 13, 12 istype 'com';

" Definitions (same as in state machine SVALETL)
...

equations

NSRED = !OVERRIDE & (LSTATE != NSGO) & (LSTATE != NSWAIT) & (LSTATE != NSWAIT2)
      # OVERRIDE & FLASHCLK;
NSYELLOW = !OVERRIDE & ((LSTATE == NSWAIT) # (LSTATE == NSWAIT2));
NSGREEN  = !OVERRIDE & (LSTATE == NSGO);

EWRED = !OVERRIDE & (LSTATE != EWGO) & (LSTATE != EWWAIT) & (LSTATE != EWWAIT2)
      # OVERRIDE & FLASHCLK;
EWYELLOW = !OVERRIDE & ((LSTATE == EWWAIT) # (LSTATE == EWWAIT2));
EWGREEN  = !OVERRIDE & (LSTATE == EWGO);

end svaletlo
```

El programa ABEL en la tabla 9-9 define solamente las variables de estado y una salida Moore registrada para el controlador de tráfico. Se necesitan seis salidas Moore para las lámparas, esto supera a las salidas disponibles de un 16V8. Por lo tanto, se emplea un PLD combinacional adicional para estas salidas, para obtener el diseño terminado que se muestra en la figura 9-6. La tabla 9-10 muestra el programa ABEL para el PLD de salida. Hemos aprovechado esta oportunidad para agregar una entrada OVERRIDE al controlador. Esta entrada puede ser asertiva por la policía para deshabilitar el controlador y poner las señales en un modo de intermitencia en rojo (cuya frecuencia está determinada por FLASHCLK), permitiéndoles despejar los embotellamientos de tráfico en forma manual gracias a este maravilloso invento.

Una máquina de estado de semáforos incluyendo la lógica de salida puede construirse en un solo 16V8, como el que se muestra en la figura 9-7, si elegimos una asignación de estado de salida codificada. Solamente se deben modificar las definiciones en el programa original de la tabla 9-9, como se ilustra en la tabla 9-11. Este PLD no incluye la entrada y el modo OVERRIDE, lo que se deja como un ejercicio (9.7).

Tabla 9-11 Definiciones para la máquina de semáforos de Sunnyvale con asignación de estado de salida codificada.

```
module svaletlb
title 'Output-Coded State Machine for Sunnyvale Traffic Lights'
SVALETLB device 'P16V8R';

" Input and output pins
CLOCK, !OE                          pin 1, 11;
NSCAR, EWCAR, TMSHORT, TMLONG       pin 2, 3, 8, 9;
NSRED, NSYELLOW, NSGREEN            pin 19, 18, 17 istype 'reg';
EWRED, EWYELLOW, EWGREEN            pin 16, 15, 14 istype 'reg';
TMRESET_L, XTRA                     pin 13, 12 istype 'reg';

" Definitions
LSTATE  = [NSRED,NSYELLOW,NSGREEN,EWRED,EWYELLOW,EWGREEN,XTRA]; " State vars
NSGO    = [   0,       0,      1,    1,       0,      0,   0]; " States
NSWAIT  = [   0,       1,      0,    1,       0,      0,   0];
NSWAIT2 = [   0,       1,      0,    1,       0,      0,   1];
NSDELAY = [   1,       0,      0,    1,       0,      0,   0];
EWGO    = [   1,       0,      0,    0,       0,      1,   0];
EWWAIT  = [   1,       0,      0,    0,       1,      0,   0];
EWWAIT2 = [   1,       0,      0,    0,       1,      0,   1];
EWDELAY = [   1,       0,      0,    1,       0,      0,   1];
```

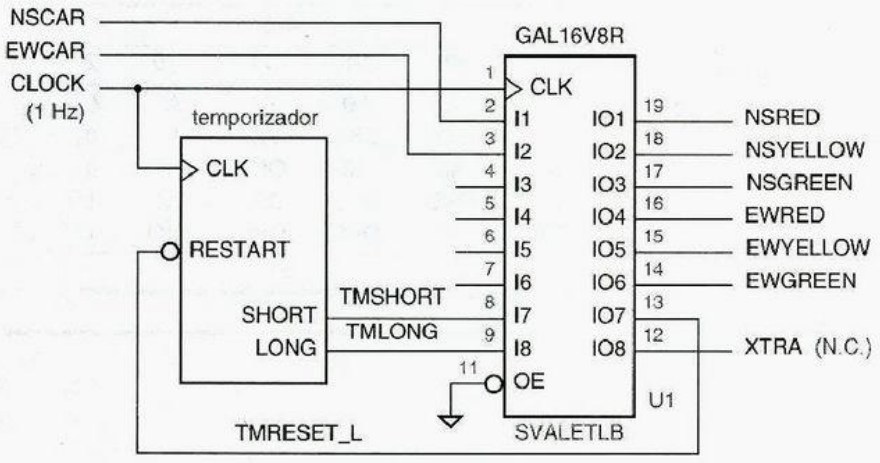


Figura 9-7
Máquina de estado de semáforos que utiliza la asignación de estado de salida codificada en un solo PLD.

9.2 Ejemplos de diseño que utilizan VHDL

Como se explicó en la sección 7.12, las características básicas del lenguaje VHDL que presentamos anteriormente en la sección 4.7, incluyendo los procesos, son poco más o menos todo lo que se necesita para modelar el comportamiento de circuito secuencial. A diferencia de ABEL, VHDL no proporciona ningún elemento de lenguaje especial para modelar máquinas de estado. En vez de ello, la mayoría de los programadores emplean una combinación de características "estándar" (principalmente los tipos numerados y los enunciados de caso "case statements") para escribir descripciones de las máquinas de estado. Emplearemos este método en los ejemplos de esta sección.

9.2.1 Ejemplos de varias máquinas simples

En la sección 7.4.1 mostramos el proceso de diseño de la tabla de estado utilizando el siguiente problema de diseño;

> Diseñe una máquina de estado sincrónica temporizada con dos entradas, A y B, y una sola salida Z, que es 1 si;
>
> - A tiene el mismo valor en cada uno de los dos anteriores tics de reloj, *o*
> - B ha sido 1 desde la última vez que la primera condición fue verdadera.
>
> De otro modo, la salida debería ser 0.

En un ambiente de diseño basado en HDL, existen muchas opciones posibles para escribir un programa que cumpla con los requerimientos establecidos, a continuación examinaremos varias.

El primer enfoque consiste en realizar una tabla de salida y estado, luego se convierte en forma manual en el programa correspondiente. Como ya hemos desarrollado una tabla de estado similar en la sección 7.4.1, ¿por qué no la utilizamos? La hemos anotado de nuevo en la tabla 9-12 y hemos escrito el correspondiente programa VHDL en la tabla 9-13.

Tabla 9-12
Tabla de salida y estado para la máquina de estado de ejemplo.

	A B				
S	***00***	***01***	***11***	***10***	***Z***
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
			S*		

Tabla 9-13
Programa VHDL para el ejemplo de la máquina de estado.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity smexamp is
  port ( CLOCK, A, B: in STD_LOGIC;
         Z: out STD_LOGIC );
end;

architecture smexamp_arch of smexamp is
type Sreg_type is (INIT, A0, A1, OK0, OK1);
signal Sreg: Sreg_type;
begin

  process (CLOCK) -- estados y transiciones de la máquina de estado
  begin
    if CLOCK'event and CLOCK = '1' then
      case Sreg is
        when INIT => if    A='0' then Sreg <= A0;
                     elsif A='1' then Sreg <= A1;   end if;
        when A0 =>   if    A='0' then Sreg <= OK0;
                     elsif A='1' then Sreg <= A1;  end if;
        when A1 =>   if    A='0' then Sreg <= A0;
                     elsif A='1' then Sreg <= OK1;  end if;
        when OK0 =>  if    A='0' then Sreg <= OK0;
                     elsif A='1' and B='0' then Sreg <= A1;
                     elsif A='1' and B='1' then Sreg <= OK1;  end if;
        when OK1 =>  if    A='0' and B='0' then Sreg <= A0;
                     elsif A='0' and B='1' then Sreg <= OK0;
                     elsif A='1' then Sreg <= OK1;            end if;
        when others => Sreg <= INIT;
      end case;
    end if;
  end process;

  with Sreg select  -- valores de salida basados en estado
    Z <= '0' when INIT | A0 | A1,
         '1' when OK0 | OK1,
         '0' when others;

end smexamp_arch;
```

Como es costumbre, la declaración de entidad VHDL especifica solamente las entradas y salidas: `CLOCK`, `A`, `B` y `Z` en este ejemplo. La definición de arquitectura especifica el funcionamiento interno de la máquina de estado. Lo primero que hace es crear un tipo enumerado, `Sreg_type`, cuyos valores son identificadores que corresponden a los nombres de estado. Luego declara una señal, `Sreg`, que se utilizará para retener el estado actual de la máquina. Debido a la manera en que `Sreg` se utiliza posteriormente, se mapeará en un registro disparado por frontera en la síntesis.

La parte de enunciados de la arquitectura tiene dos instrucciones concurrentes: una de proceso y una de asignación de selección. El proceso solamente es sensible a `CLOCK` y establece todas las transiciones de estado lo que ocurre en la frontera ascendente de `CLOCK`. Dentro del proceso, un enunciado "`if`" verifica la frontera ascendente y un enunciado `case` numera las transiciones para cada estado.

El enunciado `case` tiene seis casos, que corresponden a los cinco estados que se definieron explícitamente, y un capturador para otros estados. Por robustez, el caso "`others`" envía a la máquina de regreso al estado `INIT`. En cada caso hemos usado un enunciado anidado "`if`" para cubrir de manera explícita todas las combinaciones de entradas `A` y `B`. Sin embargo, no es estrictamente necesario incluir las combinaciones donde no hay transición del estado actual; `Sreg` retendrá su valor actual si no se hacen asignaciones en una invocación del proceso en particular.

El enunciado de asignación de selección al final de la tabla 9-13 maneja la salida de Moore de la máquina, `Z`, que es establecida a un valor como una función del estado actual. Sería fácil definir salidas Mealy dentro de esta instrucción también. Es decir, `Z` podría ser una función de las entradas además del estado actual. Ya que la instrucción "`with`" es un estado concurrente, cualquier cambio de entrada afectará la salida `Z` tan pronto como ocurra.

En realidad deberíamos haber incluido una entrada de restablecimiento en la tabla 9-13 (vea el cuadro de la página 808). Una señal `RESET` se acomoda con facilidad al modificar la declaración de entidad y agregar una cláusula a la instrucción "`if`" en la definición de arquitectura. Si `RESET` es asertiva, la máquina debería ir al estado `INIT`, de otro modo el enunciado `case` se debería ejecutar. Dependiendo de si verificamos `RESET` antes o después de la verificación del flanco de reloj, podemos crear ya sea un comportamiento de restablecimiento asincrónico o sincrónico (vea el ejercicio 9.10).

¿Qué hay cerca del problema de asignación de estado? La tabla 9-13 no indica cómo se van a asignar las combinaciones estado-variable a los estados nombrados, o incluso, ¡cuántas variables de estado se necesitan en primer lugar!

Una herramienta de síntesis puede asociar cualquier combinación binaria —o de valores enteros— deseada con los identificadores en un tipo numerado, pero una típica herramienta asignará enteros en el orden en que se numeran los nombres de estado, comenzando con 0. Posteriormente utilizará el número más pequeño posible de bits para codificar esos enteros, es decir, $\lceil \log_2 s \rceil$ bits para s estados. De este modo, el programa de la tabla 9-13 sintetizará típicamente con la misma, "más simple" asignación de estado que elegimos en el ejemplo original en la tabla 7-7. Sin embargo, VHDL soporta un par de formas en las que podemos forzar una asignación diferente.

Tabla 9-14
Uso de un atributo para forzar una codificación de enumeración.

```
library IEEE;
use IEEE.std_logic_1164.all;
library SYNOPSYS;
use SYNOPSYS.attributes.all;
...
architecture smexampe_arch of smexamp is
type Sreg_type is (INIT, A0, A1, OK0, OK1);
attribute enum_encoding of Sreg_type: type is
          "0000 0001 0010 0100 1000";
signal Sreg: Sreg_type;
...
```

Una manera de forzar una asignación es utilizar la instrucción "`attribute`" de VHDL como se muestra en la tabla 9-14. Aquí, "`enum_encoding`" es un atributo definido por el usuario cuyo valor es una cadena que especifica la codificación de enumeración que utilizará la herramienta de síntesis. El procesador del lenguaje VHDL ignora este valor, pero pasa el nombre de atributo y su valor a la herramienta de síntesis. El atributo "`enum_encoding`" es definido y conocido por la mayoría de las herramientas de síntesis, incluyendo las herramientas de SynopsYs, Inc. Cabe indicar que el programa debe utilizar un paquete "`attributes`" de Synopsys; esto es necesario para que el compilador VHDL reconozca "`enum_encoding`" como un atributo legítimo definido por el usuario. Por cierto, la asignación de estado que hemos especificado en el programa es equivalente a la codificación "casi uno-activo" de la tabla 7-7.

Otra forma de forzar una asignación, sin confiar en los paquetes externos o atributos de síntesis, es definir el registro de estado más explícitamente utilizando tipos de datos lógicos estándar. Este enfoque se muestra en la tabla 9-15. Aquí, `Sreg` está definida como un `STD_LOGIC_VECTOR` de 4 bits, y las constantes se definen para permitir que los estados sean referenciados por nombre en cualquier otra parte del programa. No se requieren otros cambios en el programa.

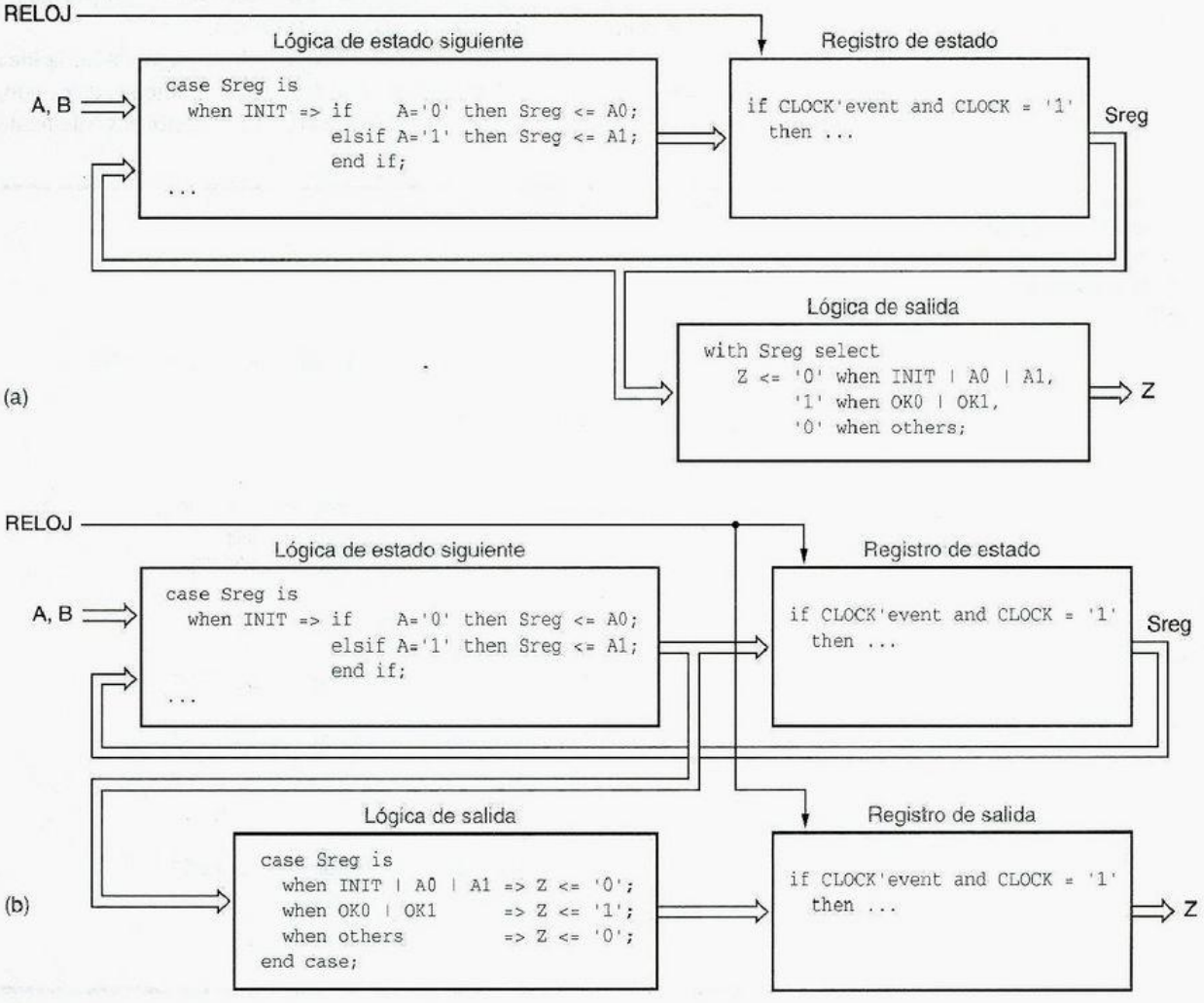
Regresando a nuestro programa VHDL original en la tabla 9-13, se puede efectuar un cambio más interesante. Como está escrito, el programa define una máquina de estado convencional tipo Moore con la estructura de la figura 9-8(a). ¿Qué sucede si conver-

Tabla 9-15
Uso de la lógica estándar y constantes para especificar una codificación de estado.

```
library IEEE;
use IEEE.std_logic_1164.all;
...
architecture smexampc_arch of smexamp is
subtype Sreg_type is STD_LOGIC_VECTOR (1 to 4);
constant INIT: Sreg_type := "0000";
constant A0  : Sreg_type := "0001";
constant A1  : Sreg_type := "0010";
constant OK0 : Sreg_type := "0100";
constant OK1 : Sreg_type := "1000";
signal Sreg: Sreg_type;
...
```

timos la instrucción de asignación de selección de la lógica de salida en un enunciado `case` y la movemos al interior del proceso de transición de estado? Al hacer esto, creamos una máquina que se podrá sintetizar con la estructura que se indica en (b). Esto es esencialmente una máquina de Mealy con salidas segmentadas cuyo comportamiento es indistinguible del comportamiento de la máquina original, excepto por la temporización. Hemos reducido el retardo de propagación desde CLOCK hasta Z al producir Z directamente en una salida de registro, pero también hemos incrementado los requerimientos de tiempo de establecimiento de A y B hasta CLOCK, debido al retardo de propagación adicional (que va de la lógica de salida hacia la entrada D del registro de salida).

Figura 9-8 Estructuras de máquina de estado implicadas mediante programas VHDL: (a) máquina de Moore con lógica de salida combinacional; (b) máquina de Mealy segmentada con registro de salida.



TEMPORIZACIÓN TRAMPOSA

Cuando escribimos la arquitectura VHDL que corresponde a la figura 9-8(b), es muy importante agregar `Sreg` a la lista de sensibilidad del proceso. Note que el enunciado `case` de la lógica de salida determina el valor de `Z` como una función de `Sreg`. Durante la primera ejecución del proceso después de una frontera de reloj ascendente, `Sreg` contiene el antiguo valor del estado. Esto es verdadero debido a que `Sreg` es una señal, no una variable. Como se explicó en la sección 4.7.9, las señales que se modifican en un proceso no adquieren sus nuevos valores hasta, por lo menos, un retardo delta *después* que se ha ejecutado el proceso. Al poner `Sreg` en la lista de sensibilidad, aseguramos que el proceso se ejecutará por segunda vez, de modo que el valor final de `Z` está basado en el nuevo valor de `Sreg`.

Todas las soluciones al ejemplo del problema de diseño de la máquina de estado que hemos mostrado hasta ahora confían en la tabla de estado que construimos originalmente en la sección 7.4.1. Sin embargo, podemos escribir directamente un programa VHDL, sin tener que elaborar en forma manual una tabla de estado.

Si nos basamos en la proposición del problema original de la página 813, la idea clave de la simplificación es eliminar el último valor de `A` de las definiciones de estado, y en lugar tener un registro por separado que le siga la pista (`LASTA`). Entonces solamente

Tabla 9-16
Máquina de estado simplificada para el problema de ejemplo VHDL.

```
architecture smexampa_arch of smexamp is
type Sreg_type is (INIT, LOOKING, OK);
signal Sreg: Sreg_type;
signal lastA: STD_LOGIC;
begin
  process (CLOCK) -- estados de la máquina de estado y transiciones
  begin
    if CLOCK'event and CLOCK = '1' then
      lastA <= A;
      case Sreg is
        when INIT    =>                    Sreg <= LOOKING;
        when LOOKING => if    A=lastA then Sreg <= OK;
                        else               Sreg <= LOOKING;
                        end if;
        when OK      => if    B='1'   then Sreg <= OK;
                        elsif A=lastA then Sreg <= OK;
                        else               Sreg <= LOOKING;
                        end if;
        when others  =>                    Sreg <= INIT;
      end case;
    end if;
  end process;

  with Sreg select  -- valores de salida basados en estado
    Z <= '1' when OK,
         '0' when others;

end smexampa_arch;
```

se deben definir dos estados no INIT (iniciales): LOOKING ("buscando todavía una coincidencia") y OK ("obtener una coincidencia o B ha sido 1 desde la última coincidencia"). La tabla 9-16 muestra una arquitectura VHDL que se fundamenta en este enfoque. En el proceso controlado por CLOCK, el primer enunciado de asignación crea el registro LASTA, y el enunciado case crea la máquina de tres estados. Al final del programa, la salida Z está definida como una simple decodificación combinacional del estado OK.

Otro ejemplo de máquina de estado simple es una "máquina de conteo de unos" con la especificación siguiente:

> Diseñe una máquina de estado sincrónica temporizada con dos entradas, X y Y, y una salida, Z. La salida debería ser 1 si el número de entradas 1 en X y Y desde el restablecimiento es un múltiplo de 4, y 0 en otro caso.

Desarrollamos una tabla de estado para esta máquina en la tabla 7-12. Sin embargo, podemos utilizar las capacidades de conteo en el paquete IEEE std_logic_arith para escribir directamente un programa VHDL para este problema.

La tabla 9-17 ilustra nuestra solución. Como siempre, existen muchas maneras diferentes de resolver el problema, y hemos seleccionado una forma que ilustra diferentes características del lenguaje. Dentro de la arquitectura, declaramos un subtipo COUNTER que es un valor UNSIGNED de 2 bits. A continuación declaramos una señal COUNT de este tipo para retener el conteo de unos, y una constante ZERO del mismo tipo para inicializar y verificar el valor de COUNT.

Tabla 9-17
Programa VHDL para una máquina de conteo de unos.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity Vonescnt is
  port ( CLOCK, RESET, X, Y: in STD_LOGIC;
         Z: out STD_LOGIC );
end;

architecture Vonescnt_arch of Vonescnt is
subtype COUNTER is UNSIGNED (1 downto 0);
signal COUNT: COUNTER;
constant ZERO: COUNTER := "00";
begin

process (CLOCK)
  begin
    if CLOCK'event and CLOCK = '1' then
      if RESET = '1' then COUNT <= ZERO;
      else COUNT <= COUNT + ('0', X) + ('0', Y);
      end if;
    end if;
  end process;

Z <= '1' when COUNT = ZERO else '0';

end Vonescnt_arch;
```

Tabla 9-18
Proceso VHDL alternativo para la máquina de conteo de unos.

```
process (CLOCK)
variable ONES: STD_LOGIC_VECTOR (1 to 2);
begin
  if CLOCK'event and CLOCK = '1' then
    ONES := (X, Y);
    if RESET = '1' then COUNT <= ZERO;
    else case ONES is
           when "01" | "10" => COUNT <= COUNT + "01";
           when "11"        => COUNT <= COUNT + "10";
           when others      => null;
         end case;
    end if;
  end if;
end process;
```

Dentro del proceso, utilizamos el método habitual para verificar un flanco ascendente en CLOCK. La cláusula "`if`" efectúa un restablecimiento sincrónico cuando se requiere, y la cláusula "`else`" agrega elegantemente 0, 1 o 2 a COUNT dependiendo de los valores de X y Y. Recuerde que una expresión como "('0', X)" es una literal del arreglo; aquí tenemos un arreglo de dos elementos STD_LOGIC, '0' y el valor actual de X. El tipo de esta literal es compatible con UNSIGNED, puesto que el número y tipo de elementos es el mismo, de modo que pueden ser combinados empleando la operación "+" definida en el paquete `std_logic_arith`. Externo al proceso, el enunciado de asignación de señal concurrente establece la salida Moore Z a 1 cuando COUNT es cero.

Para propósitos de síntesis, el enunciado "`if`" y la asignación a COUNT en la tabla 9-17 no necesariamente producen un circuito rápido o compacto. Una herramienta de síntesis sencilla, podría producir dos sumadores de 2 bits conectados en serie. Otro enfoque se ilustra en la tabla 9-18. Una herramienta inteligente puede sintetizar un incrementador más compacto para cada una de las dos sumas. De cualquier forma, la formulación de las opciones en un enunciado `case` permite que ambos sumadores o incrementadores funcionen en paralelo, y puede utilizarse un multiplexor para seleccionar una de sus salidas de acuerdo con las selecciones.

Un ejemplo final para esta subsección es la máquina de estado de latch de combinación de la sección 7.4 (a continuación omitimos la salida HINT en la especificación original):

> Diseñe una máquina de estado sincrónica temporizada con una entrada, X, y una salida, UNLK. La salida UNLK debería ser 1 si y sólo si X es 0 y la secuencia de entradas recibidas en X durante los anteriores siete tics de reloj fue 0110111.

Tabla 9-19
Programa VHDL para diseño de memoria finita de la máquina de estado de combinación con cerradura.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vcomblck is
  port ( CLOCK, RESET, X: in STD_LOGIC;
         UNLK: out STD_LOGIC );
end;

architecture Vcomblck_arch of Vcomblck is
signal XHISTORY: STD_LOGIC_VECTOR (7 downto 1);
constant COMBINATION: STD_LOGIC_VECTOR (7 downto 1) := "0110111";
begin

  process (CLOCK)
  begin
    if CLOCK'event and CLOCK = '1' then
      if RESET = '1' then XHISTORY <= "0000000";
      else XHISTORY <= XHISTORY(6 downto 1) & X;
      end if;
    end if;
  end process;

  UNLK <= '1' when (XHISTORY=COMBINATION) and (X='0') else '0';

end Vcomblck_arch;
```

Desarrollamos una tabla de estado para esta máquina en la tabla 7-14. Pero una vez más podemos tomar un enfoque diferente que es más fácil de comprender. Aquí, advertimos que la salida de la máquina en cualquier instante se encuentra completamente determinada por sus entradas durante los ocho tics precedentes de reloj. De este modo, podemos aplicar el enfoque denominado "de memoria finita" para diseñar esta máquina (vea el cuadro en la página 801). Con este enfoque, seguimos la pista de manera explícita de las siete entradas anteriores, y entonces formamos la salida como una función combinacional de estas entradas.

El programa VHDL en la tabla 9-19 es un diseño de memoria finita. Dentro de la arquitectura, el proceso meramente sigue la pista de los últimos siete valores de X utilizando lo que esencialmente es un registro de corrimiento, siendo el bit 7 el valor más antiguo de X. (Recuerde que el operador "&" en VHDL es una concatenación de arreglos.) Al exterior del proceso, la instrucción de asignación de señal concurrente establece la salida Mealy UNLK a 1 cuando X es 0 y la historia de 7 bits satisface la combinación.

9.2.2 Las luces traseras del Thunderbird

En la sección 7.5 describimos y diseñamos una máquina de estado para las "luces traseras del Ford Thunderbid". La tabla 9-20 es un programa VHDL equivalente para la máquina de las luces traseras del Thunderbird. Las transiciones de estado en esta máquina se definen exactamente del mismo modo que en el diagrama de estado de la figura 7-64. La máquina utiliza una asignación de estado codificada por salida, aprovechando que los valores de salida de las luces traseras son diferentes en cada estado.

Tabla 9-20 Programa VHDL para la máquina de las luces traseras del Thunderbird.

```
entity Vtbird is
  port ( CLOCK, RESET, LEFT, RIGHT, HAZ: in STD_LOGIC;
         LIGHTS: buffer STD_LOGIC_VECTOR (1 to 6) );
end;

architecture Vtbird_arch of Vtbird is
constant IDLE: STD_LOGIC_VECTOR (1 to 6) := "000000";
constant L3  : STD_LOGIC_VECTOR (1 to 6) := "111000";
constant L2  : STD_LOGIC_VECTOR (1 to 6) := "110000";
constant L1  : STD_LOGIC_VECTOR (1 to 6) := "100000";
constant R1  : STD_LOGIC_VECTOR (1 to 6) := "000001";
constant R2  : STD_LOGIC_VECTOR (1 to 6) := "000011";
constant R3  : STD_LOGIC_VECTOR (1 to 6) := "000111";
constant LR3 : STD_LOGIC_VECTOR (1 to 6) := "111111";
begin
  process (CLOCK)
  begin
    if CLOCK'event and CLOCK = '1' then
      if RESET = '1' then LIGHTS <= IDLE; else
        case LIGHTS is
          when IDLE => if HAZ='1' or (LEFT='1' and RIGHT='1') then LIGHTS <= LR3;
                       elsif LEFT='1'                         then LIGHTS <= L1;
                       elsif RIGHT='1'                        then LIGHTS <= R1;
                       else                                        LIGHTS <= IDLE;
                       end if;
          when L1   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= L2; end if;
          when L2   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= L3; end if;
          when L3   => LIGHTS <= IDLE;
          when R1   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= R2; end if;
          when R2   => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= R3; end if;
          when R3   => LIGHTS <= IDLE;
          when LR3  => LIGHTS <= IDLE;
          when others => null;
        end case;
      end if;
    end if;
  end process;
end Vtbird_arch;
```

PENSAMIENTOS OCIOSOS

En las máquinas de estado VHDL, no es necesario hacer una asignación explícita de un estado siguiente si se trata del mismo estado en que se encuentra la máquina. En la ejecución de un proceso, una señal VHDL conserva su valor si no se le hace ninguna asignación a ella. De esta forma, en la tabla 9-20, la cláusula final "else" podría omitirse en el estado `IDLE`, sin que modifique el comportamiento de la máquina.

Por otra parte, la operación de la máquina de estado en la tabla 9-20 podría mejorarse mediante el reemplazo de la instrucción "`null`" en el caso "`when others`" con una transición al estado `IDLE`.

9.2.3 El juego de las adivinanzas

En la sección 7.7.1 se definió una máquina de "juego de las adivinanzas" que comenzó en la página 594 con la siguiente descripción:

> Diseñe una máquina de estado sincrónica temporizada cuatro entradas, G1-G4, donde se encuentran conectados a unos botones. La máquina tiene cuatro salidas, L1-L4, que se conectan a unas lámparas o LEDs que están ubicados cerca de los botones numerados. También hay una salida ERR que se conecta a una lámpara roja. En funcionamiento normal, las salidas L1-L4 exhiben un patrón 1 de 4. En cada tic de reloj, el patrón se adelanta en una posición; la frecuencia de reloj es de aproximadamente 4 Hz.
>
> Las adivinanzas se efectúan presionando un botón, lo que produce una entrada Gi asertiva. Cuando una entrada Gi es asertiva, la salida ERR será asertiva si se presiona el botón "equivocado", es decir, si la entrada Gi que se detecta en el tic de reloj no tiene el mismo número de la salida de la lámpara que se activó antes del tic de reloj. Una vez que se hace una adivinanza, el juego se detiene y la salida ERR mantiene el mismo valor para uno o más tics de reloj hasta que la entrada Gi es negada, y luego el juego se reanuda.

Como discutimos en la sección 7.7.1, la máquina requiere seis estados: cuatro en los que la lámpara correspondiente se encuentra encendida, y dos cuando el juego se detiene después de presionar con o sin éxito un botón. En la tabla 9-21 se muestra un programa VHDL para el juego de las adivinanzas. Esta versión incluye también una entrada `RESET` para forzar al juego a un estado conocido de inicio.

El programa es en gran medida una traducción directa del diagrama de estado original de la figura 7-66. Quizás su única característica notable se encuentra en el caso "`SOK|SERR`". Puesto que las transiciones de estado siguientes para estos dos estados son idénticas (ya sea que vayan a `S1` o permanezcan en el estado actual), pueden manejarse en un caso. Sin embargo, este estilo tramposo de ahorrar tecleos no es particularmente deseable desde el punto de vista de la documentación o mantenimiento de la máquina de estado. En el caso del autor, el beneficio primario del truco era ¡ayudar a incorporar el programa en una página del libro!

Tabla 9-21 Programa VHDL para la máquina del juego de adivinanzas.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vggame is
  port ( CLOCK, RESET, G1, G2, G3, G4: in  STD_LOGIC;
         L1, L2, L3, L4, ERR:          out STD_LOGIC );
end;

architecture Vggame_arch of Vggame is
type Sreg_type is (S1, S2, S3, S4, SOK, SERR);
signal Sreg: Sreg_type;
begin

  process (CLOCK)
  begin
    if CLOCK'event and CLOCK = '1' then
      if RESET = '1' then Sreg <= SOK; else
        case Sreg is
          when S1 => if    G2='1' or G3='1' or G4='1' then Sreg <= SERR;
                     elsif G1='1'                     then Sreg <= SOK;
                     else                                  Sreg <= S2;
                     end if;
          when S2 => if    G1='1' or G3='1' or G4='1' then Sreg <= SERR;
                     elsif G1='1'                     then Sreg <= SOK;
                     else                                  Sreg <= S3;
                     end if;
          when S3 => if    G1='1' or G2='1' or G4='1' then Sreg <= SERR;
                     elsif G1='1'                     then Sreg <= SOK;
                     else                                  Sreg <= S4;
                     end if;
          when S4 => if    G1='1' or G2='1' or G3='1' then Sreg <= SERR;
                     elsif G1='1'                     then Sreg <= SOK;
                     else                                  Sreg <= S1;
                     end if;
          when SOK | SERR => if G1='0' and G2='0' and G3='0' and G4='0'
                             then Sreg <= S1; end if;
          when others => Sreg <= S1;
        end case;
      end if;
    end if;
  end process;

  L1  <= '1' when Sreg = S1   else '0';
  L2  <= '1' when Sreg = S2   else '0';
  L3  <= '1' when Sreg = S3   else '0';
  L4  <= '1' when Sreg = S4   else '0';
  ERR <= '1' when Sreg = SERR else '0';

end Vggame_arch;
```

El programa en la tabla 9-21 no especifica una asignación de estado; una máquina de síntesis típica empleará tres bits para `Sreg` y asignará los seis estados en orden para las combinaciones binarias 000-101. Para esta máquina de estado, también se puede utilizar una asignación de estado codificada por salida empleando solamente las señales de salida de error y de la lámpara. El lenguaje VHDL no proporciona un mecanismo conveniente para agrupar en conjunto las señales de salida que están en la entrada y utilizarlas para estado, sin embargo, todavía podemos obtener el efecto deseado con los cambios que se muestran en la tabla 9-22. Aquí utilizamos un comentario para documentar la correspondencia entre salidas y los bits del nuevo `Sreg` de 5 bits, y cambiamos cada una de las instrucciones de asignación de salida para seleccionar el bit apropiado en vez de decodificar completamente el estado.

Tabla 9-22 Arquitectura VHDL para el juego de adivinanzas que utiliza la asignación de estado codificada por salida.

```
architecture Vggameoc_arch of Vggame is
signal Sreg: STD_LOGIC_VECTOR (1 to 5);
-- posiciones de bit de asignación de salida codificada: L1, L2, L3, L4, ERR.
constant S1:   STD_LOGIC_VECTOR (1 to 5) := "10000";
constant S2:   STD_LOGIC_VECTOR (1 to 5) := "01000";
constant S3:   STD_LOGIC_VECTOR (1 to 5) := "00100";
constant S4:   STD_LOGIC_VECTOR (1 to 5) := "00010";
constant SERR: STD_LOGIC_VECTOR (1 to 5) := "00001";
constant SOK:  STD_LOGIC_VECTOR (1 to 5) := "00000";
begin

  process (CLOCK)
  ...              (no change to process)
  end process;

  L1  <= Sreg(1);
  L2  <= Sreg(2);
  L3  <= Sreg(3);
  L4  <= Sreg(4);
  ERR <= Sreg(5);

end Vggameoc_arch;
```

9.2.4 Reinvención de los controladores de semáforos

Si usted leyó el ejemplo ABEL de la sección 9.1.5, entonces ya me habrá escuchado despotricar sobre los espantosos controladores de semáforos en Sunnyvale, California. Realmente *parecen* estar cuidadosamente diseñados para *maximizar* el tiempo de espera de los automóviles en los cruces o intersecciones. En esta sección diseñaremos un controlador de semáforos con un comportamiento parecido al controlador de Sunnyvale.

Un cruce que se utilice con poca frecuencia (uno que tendría no más que una señal para "ceder el paso" si estuviera en Chicago) tiene los sensores y señales que se muestran

Tabla 9-23 Programa VHDL para el controlador de semáforos de Sunnyvale.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vsvale is
  port ( CLOCK, RESET, NSCAR, EWCAR, TMSHORT, TMLONG: in  STD_LOGIC;
         OVERRIDE, FLASHCLK:                          in  STD_LOGIC;
         NSRED, NSYELLOW, NSGREEN:                    out STD_LOGIC;
         EWRED, EWYELLOW, EWGREEN, TMRESET:           out STD_LOGIC );
end;

architecture Vsvale_arch of Vsvale is
type Sreg_type is (NSGO, NSWAIT, NSWAIT2, NSDELAY,
                   EWGO, EWWAIT, EWWAIT2, EWDELAY);
signal Sreg: Sreg_type;
begin

process (CLOCK)
begin
  if CLOCK'event and CLOCK = '1' then
    if RESET = '1' then Sreg <= NSDELAY; else
      case Sreg is
        when NSGO =>                                              -- Luz verde de norte a sur.
          if    TMSHORT='0' then Sreg <= NSGO;                    -- Mínimo de 5 segundos.
          elsif TMLONG='1'  then Sreg <= NSWAIT;                  -- Máximo de 5 minutos.
          elsif EWCAR='1' and NSCAR='0' then Sreg <= NSGO;        -- Hacer que el automóvil
                                                                  -- de Este a Oeste espere.
          elsif EWCAR='1' and NSCAR='1' then Sreg <= NSWAIT;      -- Basura si hay automóviles
                                                                  -- en ambas direcciones.
          elsif EWCAR='0' and NSCAR='1' then Sreg <= NSWAIT;      -- ¿Otro automóvil de Norte
                                                                  -- a Sur? ¡Deténgalo!
          else                                Sreg <= NSGO;       -- No viene ninguno,
                                                                  -- no hay cambio.
          end if;
        when NSWAIT  => Sreg <= NSWAIT2;                          -- Luz amarilla,
        when NSWAIT2 => Sreg <= NSDELAY;                          --   dos tics por seguridad.
        when NSDELAY => Sreg <= EWGO;                             -- Luz roja en ambas direc-
                                                                  --   ciones por seguridad.
        when EWGO =>                                              -- Luz verde de Este a Oeste.
        if   TMSHORT='0' then Sreg <= EWGO;                 -- Mismo comportamiento que el anterior.
          elsif TMLONG='1'  then Sreg <= EWWAIT;
          elsif NSCAR='1' and EWCAR='0' then Sreg <= EWGO;
          elsif NSCAR='1' and EWCAR='1' then Sreg <= EWWAIT;
          elsif NSCAR='0' and EWCAR='1' then Sreg <= EWWAIT;
          else                                Sreg <= EWGO;
          end if;
        when EWWAIT  => Sreg <= EWWAIT2;
        when EWWAIT2 => Sreg <= EWDELAY;
        when EWDELAY => Sreg <= NSGO;
        when others  => Sreg <= NSDELAY;                          -- Estado de "restablecimiento".
      end case;
    end if;
  end if;
end process;
```

en la figura 9-5. La máquina de estado que controla las señales de tráfico utiliza un reloj de 1 Hz y un temporizador, además tiene cuatro entradas:

NSCAR Asertiva cuando un automóvil que circula en dirección norte a sur se encuentra sobre cualquier sensor en cualquier lado del cruce.

EWCAR Asertiva cuando un automóvil que circula en dirección este a oeste se encuentra sobre cualquier sensor en cualquier lado del cruce.

TMLONG Asertiva si han pasado más de cinco minutos desde que el temporizador arrancó; permanece así hasta que se restablece el temporizador.

TMSHORT Asertiva si han pasado más de cinco segundos desde que el temporizador arrancó; permanece así hasta que se restablece el temporizador.

La máquina de estado tiene siete salidas:

NSRED, NSYELLOW y NSGREEN Controlan las luces de norte a sur.

EWRED, EWYELLOW y EWGREEN Controlan las luces de este a oeste.

TMRESET Cuando es asertiva, restablece el temporizador y niega **TMSHORT** y **TMLONG**. El temporizador inicia el cronometraje cuando **TMRESET** es negada.

Un algoritmo típico (que tiene la aprobación de las autoridades municipales para controlar los semáforos) está integrado en el programa VHDL de la tabla 9-23. Este algoritmo produce dos comportamientos que se ven con frecuencia en los semáforos "inteligentes". En la noche, cuando el tráfico es ligero, mantiene un automóvil detenido en el semáforo hasta por cinco minutos, a menos que un automóvil se aproxime en la calle transversal, en cuyo caso detiene el tráfico del cruce y deja que el automóvil que estaba esperando avance. (El sensor de "advertencia anticipada" se encuentra suficientemente atrás para cambiar las luces del semáforo antes de que el automóvil que se aproxima alcance el cruce.) Durante el día, cuando el tráfico es pesado y siempre se tienen automóviles esperando en ambas direcciones, se intercalan las señales luminosas cada cinco segundos, minimizando de este modo el uso del cruce y maximizando el tiempo de espera de todos, creando por ello un clamor generalizado para más impuestos para arreglar el problema.

Tabla 9-23 (continuación) Programa VHDL para el controlador de semáforos de Sunnyvale.

```
TMRESET  <= '1' when Sreg=NSWAIT2 or Sreg=EWWAIT2 else '0';
NSRED    <= FLASHCLK when OVERRIDE='1' else
            '1' when Sreg/=NSGO and Sreg/=NSWAIT and Sreg/=NSWAIT2 else '0';
NSYELLOW <= '0' when OVERRIDE='1' else
            '1' when Sreg=NSWAIT or Sreg=NSWAIT2 else '0';
NSGREEN  <= '0' when OVERRIDE='1' else '1' when Sreg=NSGO else '0';
EWRED    <= FLASHCLK when OVERRIDE='1' else
            '1' when Sreg/=EWGO and Sreg/=EWWAIT and Sreg/=EWWAIT2 else '0';
EWYELLOW <= '0' when OVERRIDE='1' else
            '1' when Sreg=EWWAIT or Sreg=EWWAIT2 else '0';
EWGREEN  <= '0' when OVERRIDE='1' else '1' when Sreg=EWGO else '0';

end Vsvale_arch;
```

Mientras se escribía el programa, aprovechamos la oportunidad para agregar dos entradas que no se encontraban en la especificación original. La entrada OVERRIDE puede ser asertiva por la policía para deshabilitar el controlador y poner las señales en un modo de intermitencia en rojo a una frecuencia determinada por la entrada FLASHCLK. Esto les permite solucionar en forma manual los embotellamientos de tráfico creados por esta maravillosa invención.

Tabla 9-24 Definiciones para la máquina de semáforos con asignación de estado de salida codificada.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vsvale is
  port ( CLOCK, RESET, NSCAR, EWCAR, TMSHORT, TMLONG: in  STD_LOGIC;
         OVERRIDE, FLASHCLK:                          in  STD_LOGIC;
         NSRED, NSYELLOW, NSGREEN:                    out STD_LOGIC;
         EWRED, EWYELLOW, EWGREEN, TMRESET:           out STD_LOGIC );
end;

architecture Vsvaleoc_arch of Vsvale is
signal Sreg: STD_LOGIC_VECTOR (1 to 7);
-- posiciones de bits de asignación de salida codificada:  (1) NSRED, (2) NSYELLOW,
--                  (3) NSGREEN, (4) EWRED, (5) EWYELLOW, (6) EWGREEN, (7) EXTRA
constant NSGO:    STD_LOGIC_VECTOR (1 to 7) := "0011000";
constant NSWAIT:  STD_LOGIC_VECTOR (1 to 7) := "0101000";
constant NSWAIT2: STD_LOGIC_VECTOR (1 to 7) := "0101001";
constant NSDELAY: STD_LOGIC_VECTOR (1 to 7) := "1001000";
constant EWGO:    STD_LOGIC_VECTOR (1 to 7) := "1000010";
constant EWWAIT:  STD_LOGIC_VECTOR (1 to 7) := "1000100";
constant EWWAIT2: STD_LOGIC_VECTOR (1 to 7) := "1000101";
constant EWDELAY: STD_LOGIC_VECTOR (1 to 7) := "1001001";

begin

process (CLOCK)
...             (no change to process)
end process;

TMRESET  <= '1' when Sreg=NSWAIT2 or Sreg=EWWAIT2 else '0';
NSRED    <= Sreg(1);
NSYELLOW <= Sreg(2);
NSGREEN  <= Sreg(3);
EWRED    <= Sreg(4);
EWYELLOW <= Sreg(5);
EWGREEN  <= Sreg(6);

end Vsvaleoc_arch;
```

Como sucede en la mayor parte de nuestros ejemplos, la tabla 9-23 no proporciona una asignación específica de estado. Y como muchos de nuestros ejemplos, esta máquina de estado funcionará bien con una asignación de estado codificada por salida. Muchos de los estados pueden ser identificados mediante una combinación única de valores de salida de luces. Pero hay tres pares de estados que no son distinguibles al examinar únicamente las luces: (NSWAIT, NSWAIT2), (EWWAIT, EWWAIT2) y (NSDELAY, EQDELAY). Podemos manejarlos agregando una variable de estado adicional, "EXTRA", que tiene diferentes valores para los dos estados en cada par. Esta idea es realizada en el programa modificado de la tabla 9-24.

Ejercicios

9.1 Escriba un programa ABEL para la máquina de estado descrita en el ejercicio 7.30.

9.2 Modifique el programa ABEL de la tabla 9-2 para incluir la salida HINT de la especificación original de la máquina de estado en la sección 7.4.

9.3 Vuelva a diseñar la máquina de las luces traseras del Thunderbird de la sección 9.1.3 para incluir funciones de luces de estacionamiento y luces de frenos. Cuando la entrada BRAKE ("freno") sea asertiva, todas las luces deberían encenderse inmediatamente, y permanecer así hasta que BRAKE sea negada, independientemente de cualquier otra función. Cuando la entrada PARK ("estacionar") sea asertiva, cada lámpara se enciende a un 50% de su brillo todo el tiempo cuando de otro modo estarían apagadas. Esto se consigue al controlar la lámpara con una señal DIMCLK de 100 Hz con un 50% del ciclo de trabajo. Dibuje un diagrama lógico para el circuito utilizando uno o dos PLD, escriba un programa ABEL para cada PLD y escriba una descripción breve de cómo funciona su sistema.

9.4 Encuentre una asignación de estado de 3 bits para la máquina del juego de las adivinanzas en la tabla 9-5 que reduzca el número máximo de términos de producto por salida a 7. ¿Puede usted hacerlo aún mejor?

9.5 El funcionamiento del juego de las adivinanzas en la sección 9.1.4 es fácil de predecir; es fácil para un jugador aprender la velocidad a la cual cambian las luces y presionar siempre el botón en el momento correcto. El juego es más divertido si la velocidad de cambio es más variable.

Modifique la máquina de estado ABEL en la tabla 9-5 de manera que en los estados S1–S4, la máquina avance solamente si una nueva entrada, SEN, es asertiva. (SEN está destinada a estar conectada a un generador de flujo de bits pseudoaleatorio.) Deberían ser reconocidos tanto los éxitos como los fracasos al oprimir los botones, ya sea que SEN sea asertiva o no. Determine si su diseño modificado todavía cabe en un 16V8.

9.6 En relación con el ejercicio anterior, escriba un programa ABEL para un LFSR de 8 bits que utiliza un 16V8, de tal modo que una de sus salidas pueda emplearse como un generador de flujo de bits pseudoaleatorio. ¿Después de cuántos tics de reloj se repite la secuencia de bits? ¿Cuál es el número máximo de ceros que se presentan en un renglón? ¿Y de unos?

9.7 Agregue una entrada OVERRIDE a la máquina de estado de los semáforos de la figura 9-7, utilizando todavía un 16V8 solo. Cuando OVERRIDE es asertiva, las luces rojas deberían encenderse y apagarse de forma intermitente, cambiando una vez por segundo. Escriba un programa ABEL completo para su máquina.

9.8 Modifique el comportamiento de la máquina del controlador de semáforos ABEL en la tabla 9-9 para tener un comportamiento más razonable, el tipo de comportamiento que a usted le gustaría ver en los semáforos de su propia ciudad.

9.9 Escriba un programa VHDL para la máquina de estado descrita en el ejercicio 7.30.

9.10 Muestre cómo modificar la tabla 9-13 para proporcionar una entrada asincrónica `RESET` que obligue a la máquina de estado a ir al estado `INIT`. Repita lo anterior para una versión sincrónica que obligue a la máquina de estado a ir al estado `INIT` si la señal de `RESET` es asertiva en la frontera ascendente del reloj.

9.11 Escriba y pruebe un programa VHDL correspondiente a la figura 9-8(b). Utilizando una herramienta de síntesis disponible, determine si el circuito resultante de esta versión difiere del resultante de la figura 9-8(a), y cómo difiere.

9.12 Escriba un programa VHDL para la máquina de estado del contador de unos como se describió mediante la tabla de estado en la tabla 7-12.

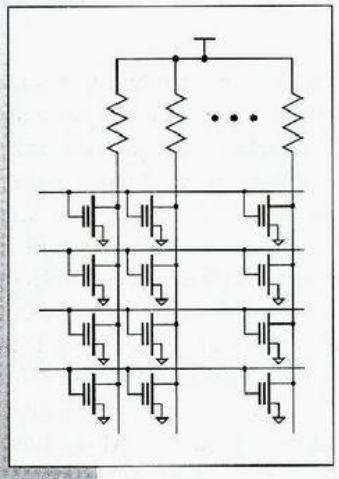
9.13 Escriba un programa VHDL para la máquina de estado de combinación de cerradura como se describió mediante la tabla de estado en la tabla 7.14.

9.14 Modifique el programa VHDL de la tabla 9-19 para incluir la salida HINT de la especificación original de la máquina de estado en la sección 7.4.

9.15 Repita el ejercicio 9.3 empleando VHDL, suponiendo que está dirigiendo su diseño a un CPLD o FPGA simple.

9.16 Modifique el programa VHDL del juego de las adivinanzas de la tabla 9-21 de acuerdo a la idea del ejercicio 9.5. Agregue otro proceso al programa para proporcionar un generador de flujo de bits pseudoaleatorio de acuerdo al ejercicio 9.6.

9.17 Modifique el comportamiento de la máquina del controlador de semáforos VHDL en la tabla 9-23 para tener un comportamiento más razonable, la clase de comportamiento que a usted le gustaría tener en los semáforos de su propia ciudad.



capítulo 10

Memorias, dispositivos CPLD y FPGA

Cualquier circuito secuencial tiene memoria de una clase, puesto que cada flip-flop o latch almacena un bit de información. Sin embargo, por lo regular reservamos la palabra "memoria" para hacer referencia a los bits que se almacenan en una forma estructurada, habitualmente como un arreglo bidimensional en el cual se tiene acceso a un renglón de bits a la vez.

En este capítulo se describen varios tipos diferentes de organizaciones de memoria y chips de memoria disponibles comercialmente. Las mismas clases de memoria pueden estar integradas en chips VLSI más grandes, donde se combinan con otros circuitos para efectuar una función útil.

Las aplicaciones de la memoria son muchas y muy variadas. En una unidad de procesamiento central (CPU, central processing unit) de un microprocesador, una "memoria de sólo lectura" puede ser empleada para definir los pasos primitivos que se llevarán a cabo para ejecutar instrucciones en el conjunto de instrucciones de la CPU. Al lado de la CPU, una "memoria estática" rápida puede funcionar como una memoria caché (memoria de almacenamiento intermedio temporal) para mantener los datos e instrucciones recientemente utilizados. Y un subsistema de memoria principal de un microprocesador puede contener cientos de millones de bits en "memoria dinámica" que almacenan sistemas operativos completos, programas y datos.

Las aplicaciones de la memoria no están limitadas a microprocesadores o inclusive a sólo sistemas digitales. Por ejemplo, el equipo en el sistema de telefonía pública utiliza memorias de sólo lectura para efectuar ciertas transformaciones en señales de voz digitalizada, y "memorias estáticas" rápidas como una "fábrica de conmutación" para enrutar la voz digitalizada entre los suscriptores. Muchos reproductores

portátiles de discos compactos de audio "adelantan la lectura" y almacenan unos cuantos segundos de audio en una "memoria dinámica" de modo que la unidad puede mantenerse reproduciendo el audio incluso si es sacudido físicamente (esto requiere de más o menos 1.4 millones de bits por segundo de audio almacenado). Y existen muchos ejemplos de equipo audiovisual moderno que utiliza memorias para almacenar de manera temporal señales digitalizadas para mejorar a través del procesamiento de señales digitales.

Este capítulo comienza con una discusión sobre memoria de sólo lectura y sus aplicaciones y luego describe dos diferentes tipos de memoria de lectura/escritura. Las secciones opcionales describen la estructura interna de los diferentes tipos de memoria.

En las últimas dos secciones de este capítulo se analizan los CPLD y FPGA. Estos dispositivos son semejantes a las memorias en el sentido de que son estructuras grandes y regulares que pueden utilizarse en gran variedad de aplicaciones. Al habilitar un desarrollo muy rápido de funciones lógicas personalizadas, han llegado a convertirse en bloques de construcción esenciales en el diseño digital moderno.

10.1 Memoria de sólo lectura

memoria de sólo lectura
entrada de dirección
salida de datos

Una *memoria de sólo lectura* (*ROM, read-only memory*) es un circuito combinacional con n entradas y b salidas, como se ilustra en la figura 10-1. Las entradas se llaman *entradas de dirección* y se denominan de manera tradicional A0, A1, ... , An–1. Las salidas se conocen como *salidas de datos* y habitualmente se denominan D0, D1, ... , Db-1.

Una ROM "almacena" la tabla de verdad de una función lógica combinacional de n entradas y b salidas. Por ejemplo, la tabla 10-1 es la tabla de verdad de una función combinacional de 3 entradas y 4 salidas; y podría almacenarse en una ROM de $2^3 \times 4$ (8×4). Despreciando los retardos de propagación, las salidas de datos de una ROM igualan todas las veces los bits de salida en el renglón de la tabla de verdad de selección por las entradas de dirección.

Puesto que una ROM es un circuito combinacional, usted estaría en lo correcto al decir que no es en realidad una memoria en absoluto. En términos del funcionamiento de un circuito digital, se puede tratar a una memoria ROM como si fuera cualquier otro ele-

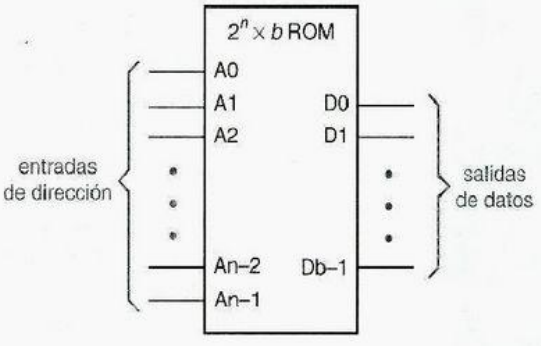


Figura 10-1
Estructura básica de una ROM de $2^n \times b$.

MEMORIA: ¡NO! La mayoría de los tipos de memoria sólo de lectura no son en realidad memoria en el sentido más estricto de la palabra, porque son circuitos combinacionales, no secuenciales. Se conocen como "memoria" debido al paradigma organizacional que describe su función.

Entradas			Salidas			
A2	A1	A0	D3	D2	D1	D0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Tabla 10-1
Tabla de verdad para una función lógica combinacional de 3 entradas, 4 salidas.

mento lógico combinacional. Sin embargo, también se puede pensar que la información se "almacenará" en la ROM cuando se fabrica o programa (en la sección 10.1.4 indicaremos cómo se puede realizar esto).

Aunque pensemos en la ROM como si fuera un tipo de memoria, tiene una importante diferencia en comparación con la mayoría de los tipos de memoria de circuitos integrados, la ROM es una *memoria no volátil*; es decir, su contenido se conserva incluso si no se aplica energía alguna.

memoria no volátil

10.1.1 Uso de las memorias ROM para funciones lógicas combinacionales "aleatorias"

La tabla 10-1 es en realidad la tabla de verdad de un decodificador de 2 a 4 con un control de polaridad de salida, una función que puede ser construida con compuertas discretas como se ilustra en la figura 10-2. De este modo, tenemos dos maneras diferentes para construir el decodificador: con compuertas discretas o con un reloj de 8×4 que contenga la tabla de verdad, como se ilustra en la figura 10-3.

El patrón de asignación de las entradas y salidas del decodificador para las entradas y salidas de la ROM en la figura 10-3 es una consecuencia de la manera en que la tabla de

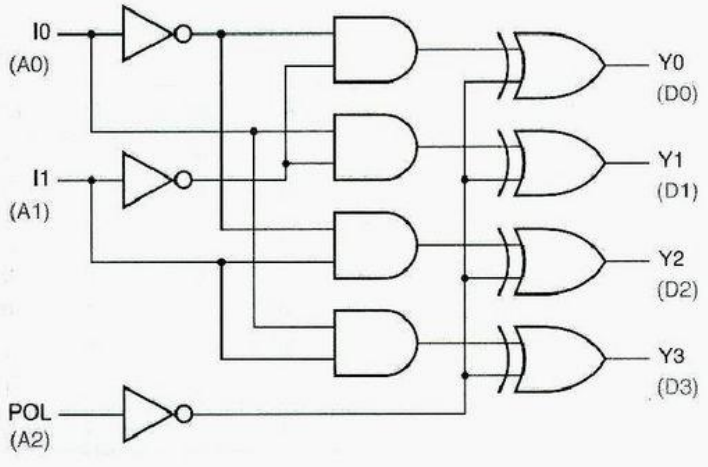


Figura 10-2
Un decodificador de 2 a 4 con control de polaridad de salida.

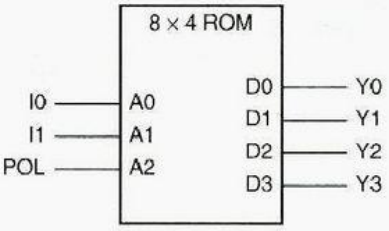


Figura 10-3
Conexiones para construir el decodificador de 2 a 4 utilizando una ROM de 8 × 4 donde se almacena la tabla 10-1.

verdad se construyó en la tabla 10-1. De este modo, la realización física de la memoria ROM del decodificador no es única. Esto es, podríamos escribir los renglones o columnas de la tabla de verdad en un orden diferente y utilizar una ROM físicamente distinta para realizar la misma función lógica, tan sólo mediante la asignación de las señales del decodificador a entradas y salidas diferentes de la ROM. Otra manera de ver esto es que podemos renombrar las entradas de dirección y las salidas de datos individuales de la ROM.

Por ejemplo, intercambiando los bits en las columnas D0 y D3 de la tabla 10-1 podríamos obtener la tabla de verdad para una ROM físicamente diferente. Sin embargo, la nueva ROM todavía podría ser utilizada para construir el decodificador de 2 a 4 simplemente intercambiando las etiquetas Y0 y Y3 en la figura 10-3. Del mismo modo, si mezclamos los renglones de datos de la tabla de verdad como se muestra en la tabla 10-2, obtendríamos otra ROM diferente, pero todavía podría utilizarse como decodificador de 2 a 4 con una reorganización de las entradas de dirección, A0 = POL, A1 = I0, A2 = I1.

Cuando se construye una ROM para almacenar una tabla de verdad dada, las señales de entrada y salida que se leen de derecha a izquierda en la tabla de verdad se asignan por lo normal a las entradas de dirección y salidas de datos de la ROM con etiquetas ascendentes. Cada combinación de dirección o datos puede leerse entonces como un entero binario correspondiente con los bits numerados en forma "natural". Un archivo de datos se utiliza casi siempre para especificar la tabla de verdad que se almacenará en la ROM cuando es fabricada o programada. El archivo de datos por lo común proporciona los valores de dirección y datos como números hexadecimales. Por ejemplo, un archivo de datos puede especificar la tabla 10-2 al decir qué direcciones 0-7 de la ROM deberían almacenar los valores E, 1, D, 2, B, 4, 7, 8.

Tabla 10-2
Tabla de verdad con renglones de datos mezclados.

Entradas			*Salidas*			
A2	***A1***	***A0***	***D3***	***D2***	***D1***	***D0***
0	0	0	1	1	1	0
0	0	1	0	0	0	1
0	1	0	1	1	0	1
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	1	0	0	1	1	1
1	1	1	1	0	0	0

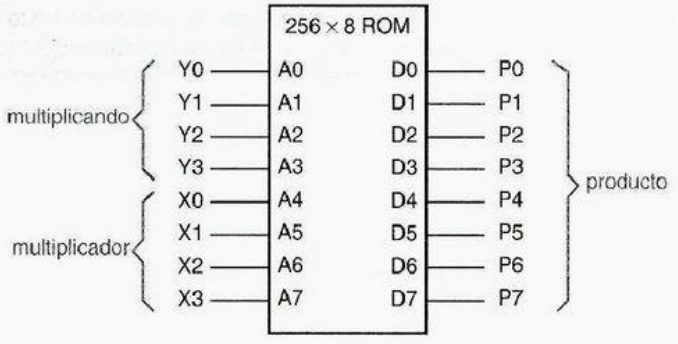


Figura 10-4 Conexiones para efectuar una multiplicación binaria sin signo de 4 × 4 empleando una ROM de 256 × 8.

Otro ejemplo simple de una función que puede ser construida con una ROM es una multiplicación binaria sin signo de 4×4. Mostramos un programa ABEL para esta función en la sección 5.11.2 y hallamos que el número de términos de producto requeridos (36) era demasiado alto para obtenerlo con sólo una pasada a través de un arreglo AND-OR de un PLD convencional. A su vez, podemos realizar la función en un paso a través de una ROM de $2^8 \times 8$ (256×8) con las conexiones mostradas en la figura 10-4.

El contenido de una ROM normalmente se especifica mediante un archivo que contiene una entrada para cada dirección en la ROM. Por ejemplo, la tabla 10-3 es un listado hexadecimal del contenido de la ROM multiplicadora de 4×4. Cada renglón proporciona una dirección de partida en la ROM y especifica los valores de datos de 8 bits almacenados en 16 direcciones sucesivas.

El punto satisfactorio acerca del diseño basado en ROM es que usted puede por lo regular escribir un programa simple en un lenguaje de alto nivel para calcular qué debería ser almacenado en la ROM. Por ejemplo, nos lleva sólo unos cuantos minutos escribir un programa en C, mostrado en la tabla 10-4, que ha generado el contenido de la tabla 10-3.

Tabla 10-3 Archivo de texto hexadecimal especificando el contenido de una ROM de multiplicador 4 × 4.

```
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
20: 00 02 04 06 08 0A 0C 0E 10 12 14 16 18 1A 1C 1E
30: 00 03 06 09 0C 0F 12 15 18 1B 1E 21 24 27 2A 2D
40: 00 04 08 0C 10 14 18 1C 20 24 28 2C 30 34 38 3C
50: 00 05 0A 0F 14 19 1E 23 28 2D 32 37 3C 41 46 4B
60: 00 06 0C 12 18 1E 24 2A 30 36 3C 42 48 4E 54 5A
70: 00 07 0E 15 1C 23 2A 31 38 3F 46 4D 54 5B 62 69
80: 00 08 10 18 20 28 30 38 40 48 50 58 60 68 70 78
90: 00 09 12 1B 24 2D 36 3F 48 51 5A 63 6C 75 7E 87
A0: 00 0A 14 1E 28 32 3C 46 50 5A 64 6E 78 82 8C 96
B0: 00 0B 16 21 2C 37 42 4D 58 63 6E 79 84 8F 9A A5
C0: 00 0C 18 24 30 3C 48 54 60 6C 78 84 90 9C A8 B4
D0: 00 0D 1A 27 34 41 4E 5B 68 75 82 8F 9C A9 B6 C3
E0: 00 0E 1C 2A 38 46 54 62 70 7E 8C 9A A8 B6 C4 D2
F0: 00 0F 1E 2D 3C 4B 5A 69 78 87 96 A5 B4 C3 D2 E1
```

Tabla 10-4 Programa para generar el archivo de texto especificando el contenido de una ROM de multiplicador de 4 × 4.

```
#include <stdio.h>

/* Procedure to print d as a hex digit. */
void PrintHexDigit(int d)
{
    if (d<10) printf("%c", '0'+d);
    else printf("%c", 'A'+d-10);
}

/* Procedure to print i as two hex digits. */
void PrintHex2(int i)
{
    PrintHexDigit((i / 16) % 16);
    PrintHexDigit(i % 16);
}

void main()
{
  int x, y;

  for (x=0; x<=15; x++) {
    PrintHex2(x*16); printf(":");
    for (y=0; y<=15; y++) {
      printf(" ");
      PrintHex2(x*y);
    }
    printf("\n");
  }
}
```

*10.1.2 Estructura interna de la ROM

El mecanismo utilizado por las ROM para "almacenar" información varía con las diferentes tecnologías ROM. En la mayoría de las ROM, la presencia o ausencia de un diodo o transistor es la distinción entre un 0 y un 1.

La figura 10-5 es el esquema de una ROM primitiva de 8 × 4 que usted mismo podría construir utilizando un decodificador MSI y un puñado de diodos. Las entradas de dirección seleccionan una de las salidas del decodificador que van a ser asertivas. Cada salida del decodificador se denomina una *línea de palabra* debido a que selecciona un renglón o palabra de la tabla almacenada en la ROM. La figura muestra la situación con A2–A0 = 101 y ROW5_L afirmada.

línea de palabra

línea de bit

Cada línea vertical en la figura 10-5 se conoce como una *línea de bit* porque corresponde a un bit de salida de la ROM. Una línea de palabra asertiva lleva a una línea de bit al nivel BAJO si un diodo se encuentra conectado entre la línea de palabra y la línea de

*A lo largo de este libro, las secciones opcionales están marcadas con un asterisco.

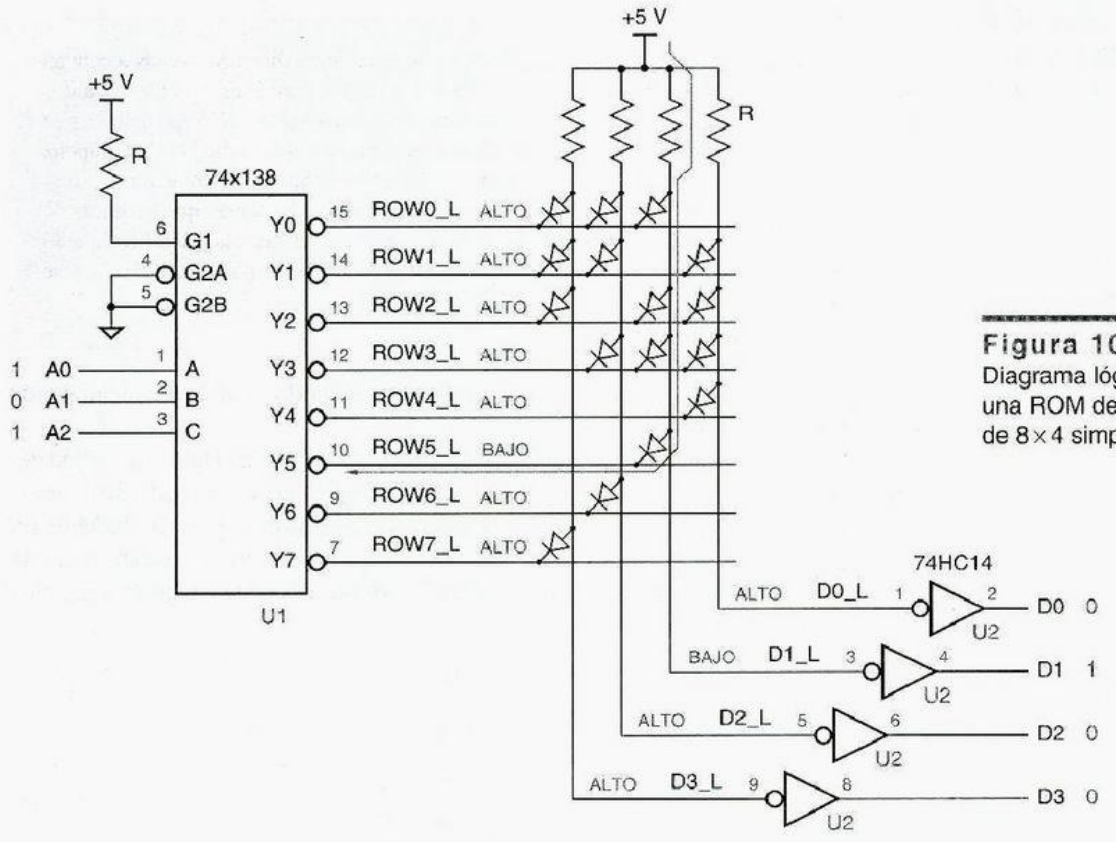


Figura 10-5
Diagrama lógico de una ROM de diodos de 8×4 simple.

bit. Existe sólo un diodo en el renglón 5, y la correspondiente línea de bit (D1_L) se lleva al nivel BAJO. Las líneas de bit son almacenadas temporalmente a través de inversores para producir las salidas de ROM D3-D0, 0010 para el caso mostrado.

En el circuito ROM de la figura 10-5, cada intersección entre una línea de palabras y una línea de bits corresponde a un bit de "memoria". Si un diodo está presente en la intersección, se almacena un 1; de otro modo, se almacena un 0. Si se fuera a construir este circuito en el laboratorio, se "programaría" la memoria insertando y retirando diodos en cada intersección. Aunque parezca primitivo, los propietarios de la minicomputadora DEC PDP-11 (hacia 1970) utilizaban una tecnología similar en el "módulo ROM de carga de sistema" M792 32×16. El módulo fue embarcado con 512 diodos soldados en

DETALLES, DETALLES Los inversores con umbrales de entrada CMOS son utilizados en la figura 10-5 para mejorar los márgenes de ruido del circuito, debido a que la caída de voltaje de 0.7 volts a través de los diodos crea un nivel BAJO en las líneas de bit que en realidad no es tan bajo. Por fortuna, con el 74HC04, un nivel BAJO es algo menor que 1.35 V. Por supuesto, usted nunca construiría este circuito excepto en un laboratorio académico: es más fácil adquirir y programar un chip de ROM comercial.

TRAYECTORIAS FURTIVAS

El circuito ROM de la figura 10-5 debe utilizar diodos, no conexiones directas, en cada localidad donde vaya a almacenarse un 1. La figura 10-6 muestra lo que ocurre si apenas unos cuantos diodos (como los del renglón 3) son reemplazados con conexiones directas. Supongamos que las entradas de dirección son 101; entonces ROW5_L es asertiva, y solamente D1_L se supone que se llevará al nivel BAJO para crear una salida de 0010. Sin embargo, las conexiones directas permiten que fluya corriente a lo largo de *trayectorias furtivas*, de manera que las líneas de bit D2_L y D0_L también son llevadas al nivel BAJO, y la salida (incorrecta) es 0111. Cuando se utilizan diodos, las trayectorias furtivas son bloqueadas por diodos de polaridad inversa, y se obtienen las salidas correctas.

su lugar, y los propietarios lo programaban recortando el diodo en cada localidad donde un 0 estaba por ser almacenado.

El patrón de diodos mostrado en la figura 10-5 corresponde a la tabla de verdad del decodificador de 2 a 4 de la tabla 10-1. Esto no parece ser muy eficiente (utilizamos un decodificador de 3 a 8 y un manojo de diodos para construir la versión ROM de un decodificador de 2 a 4). ¡Podríamos haber empleado un subconjunto del decodificador de 3 a 8 directamente! Mostraremos una estructura ROM más eficiente y algunos ejemplos de diseño más útiles más adelante.

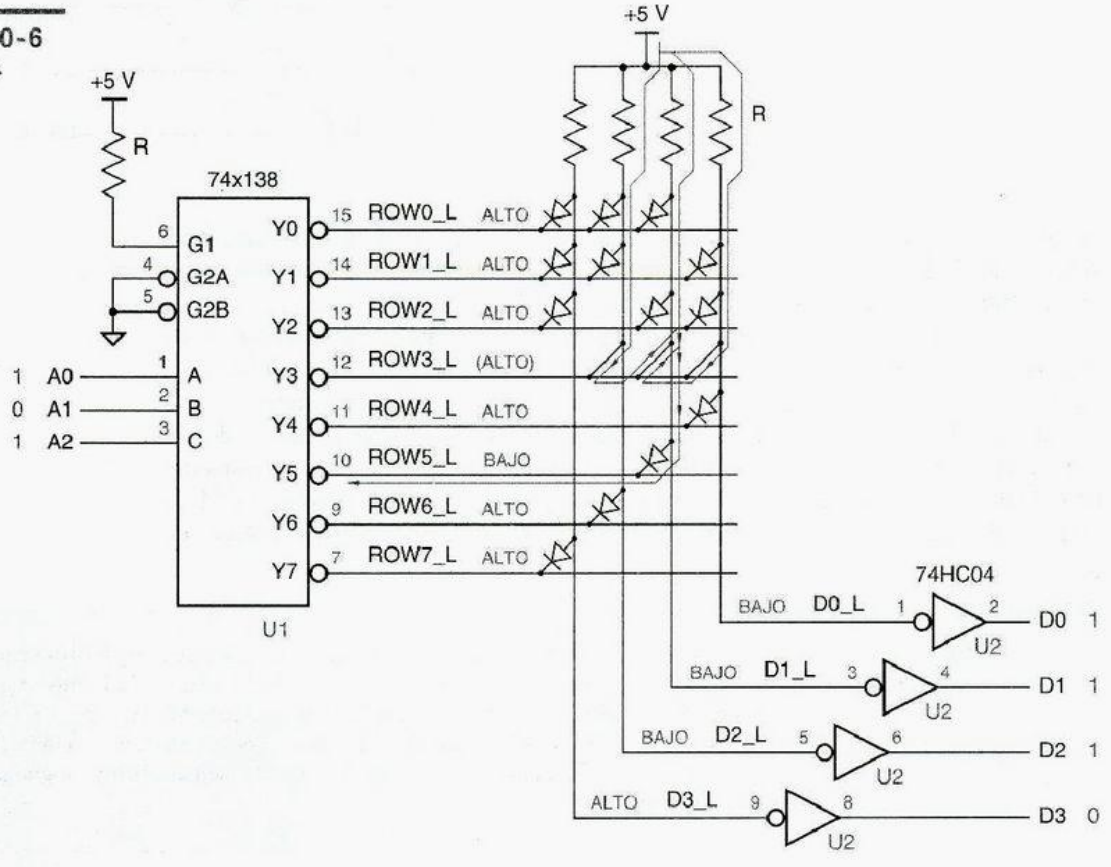


Figura 10-6
Trayectorias furtivas en una ROM.

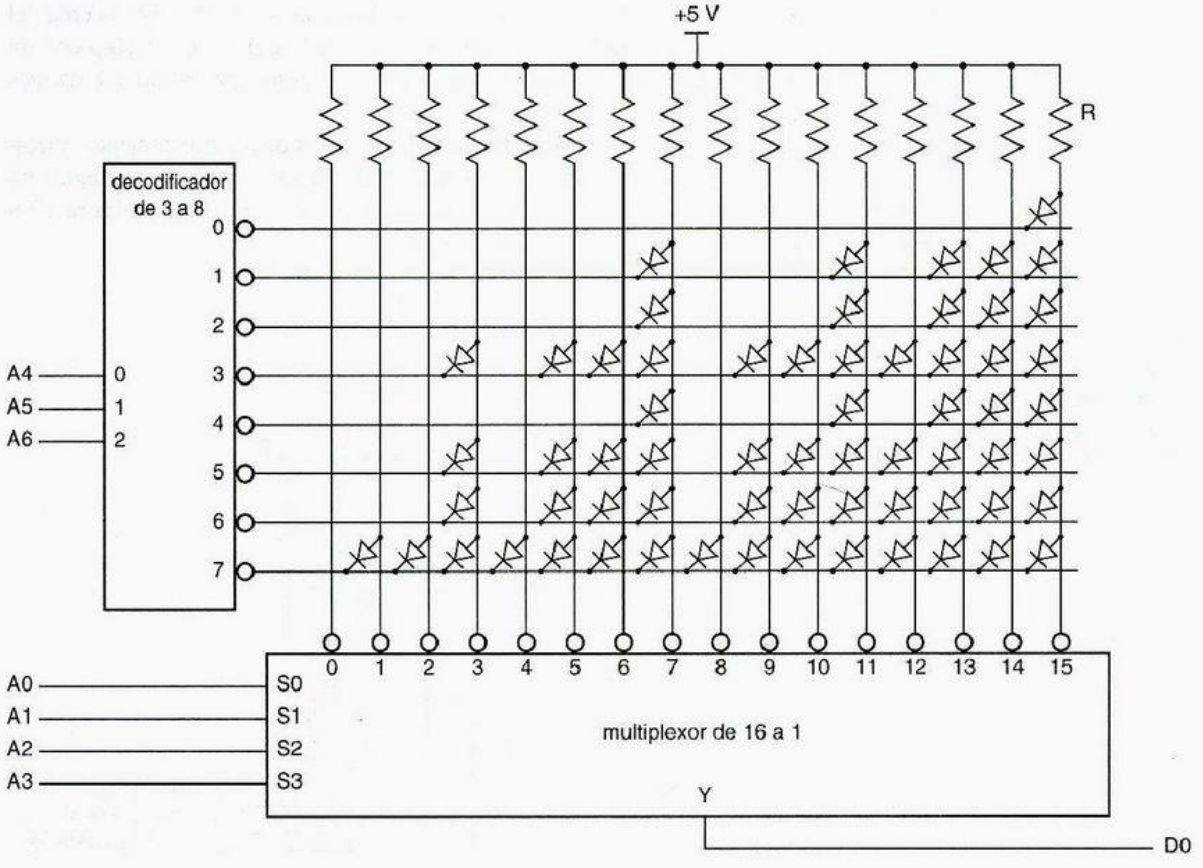
*10.1.3 Decodificación bidimensional

Supongamos que usted quiere construir una ROM de 128 × 1 utilizando la clase de estructura descrita en la subsección anterior. ¿Alguna vez ha pensado sobre lo que le tomaría construir un decodificador de 7 a 128? ¡Intente comenzar con 128 compuertas NAND de 7 entradas y agregue 14 buffers e inversores con una cargabilidad de salida de 64 cada uno! Se pueden encontrar ROM comerciales con un millón de bits o más; aunque resulte increíble, no contienen decodificadores 20 a 1,048,576. En su lugar, se emplea una estructura diferente, denominada *decodificación bidimensional*, para reducir las dimensiones del decodificador a una cantidad del orden de la raíz cuadrada del número de direcciones.

decodificación bidimensional

La idea básica en la decodificación bidimensional es acomodar las celdas de la ROM en un arreglo que se encuentre tan cerca como sea posible del cuadrado. Por ejemplo, la figura 10-7 muestra una posible estructura interna para una ROM de 128 × 1. Los tres bits de dirección de orden superior, A6–A4, se utilizan para seleccionar un renglón. Cada renglón almacena 16 bits comenzando en la dirección (A6, A5, A4, 0, 0, 0, 0). Cuando una dirección se aplica a la memoria ROM, los 16 bits del renglón seleccionado se "extraen" en paralelo en las líneas de bits. Un multiplexor de 16 entradas selecciona el bit de datos deseado basado en los bits de dirección de orden inferior.

Figura 10-7 Estructura interna de una ROM de 128 × 1 que utiliza una decodificación bidimensional.



TRANSISTORES COMO ELEMENTOS DE ROM

Las ROM MOS en realidad utilizan un transistor en lugar de un diodo en cada localidad donde va a almacenarse un bit; la figura 10-8 ilustra la idea básica. El decodificador de renglón tiene salidas activas en estado alto. Cuando una línea de renglón es asertiva, los transistores NMOS en ese renglón se encienden, lo que lleva a las líneas de bit correspondientes al nivel bajo. Una idea semejante puede emplearse en ROM construidas con transistores bipolares.

Por cierto, el patrón de diodos en la figura 10-7 no fue elegido al azar. Efectúa una función lógica combinacional de 7 entradas muy útil que requeriría 35 compuertas AND de 4 entradas para construirse como un circuito AND-OR de dos niveles mínimo (véase el ejercicio 10.7). La versión ROM de esta función en realidad ahorra bastante cantidad de esfuerzo de ingeniería y espacio en un diseño al nivel de tarjeta.

La decodificación bidimensional permite que se construya una ROM de 128×1 con un decodificador de 3 a 8 y un multiplexor de 16 bits (cuya complejidad es comparable a la de un decodificador de 4 a 16). Podría construirse una ROM de $1M \times 1$ con un decodificador de 10 a 1024 y un multiplexor de 1024 entradas; algo nada fácil, pero mucho más simple que la alternativa unidimensional.

Aparte de reducir la complejidad de decodificación, la decodificación bidimensional tiene otro beneficio: conduce a un chip cuyas dimensiones físicas están cerca del cuadrado, algo importante para la fabricación y encapsulado del chip. Un chip con un arreglo físico de $1M \times 1$ sería *muy* largo y delgado y no podría ser construido por razones económicas.

En las ROM con múltiples salidas de datos, los arreglos de almacenamiento correspondientes a cada salida de datos pueden hacerse más estrechos a fin de conseguir un esquema de chip total que esté más cercano al cuadrado. Por ejemplo, la figura 10-9 muestra el esquema posible para una ROM de $32K \times 8$.

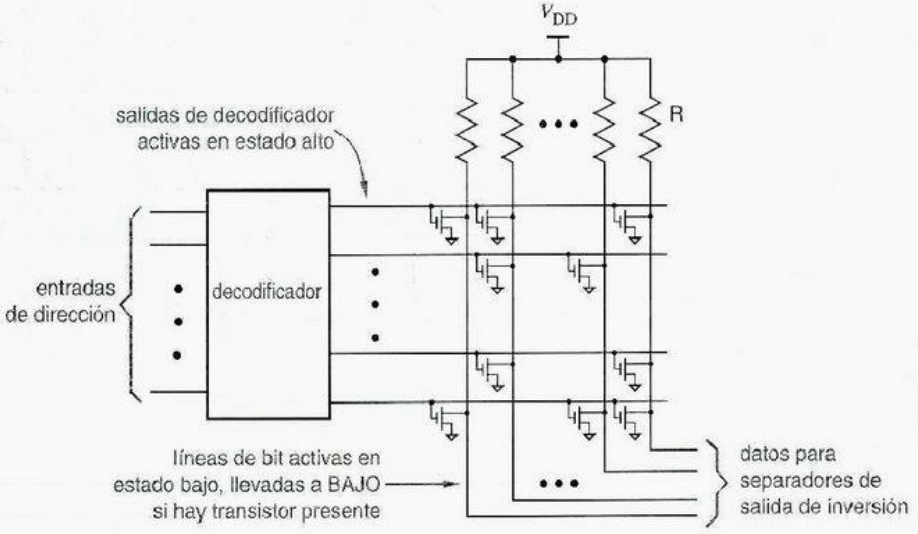


Figura 10-8
Transistores MOS como elementos de almacenamiento en una ROM.

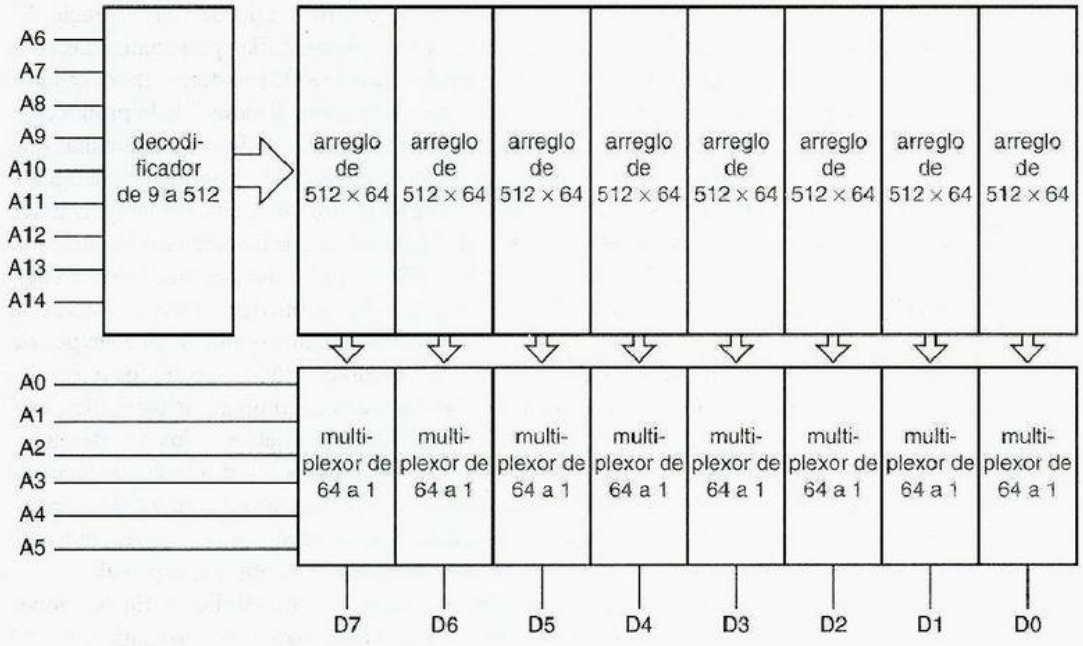


Figura 10-9 Posible composición de una ROM de 32K × 8.

10.1.4 Tipos comerciales de ROM

A menos que usted visite museos de computadoras, es imposible en la actualidad encontrar módulos ROM construidos con diodos discretos. Una ROM moderna está fabricada como un simple chip de CI; uno que almacena cuatro megabits tiene un precio aproximado inferior a los $5 USD. Se utilizan diversos métodos para "programar" la información almacenada en una ROM, como se estudia a continuación y se resume en la tabla 10-5.

ROM de plantilla programable
ROM de plantilla plantilla

La mayor parte de las primeras memorias ROM del tipo circuito integrado eran memorias *ROM de plantilla programable* (o simplemente, *ROM de plantilla*). Una ROM de plantilla es programada por el patrón de conexiones y por la ausencia de conexiones en una de las *plantillas* utilizadas en el proceso de fabricación del CI. Para programar o escribir información en la ROM, el cliente proporciona al fabricante un listado del con-

Tabla 10-5 Tipos comerciales de ROM.

Tipo	*Tecnología*	*Ciclo de lectura*	*Ciclo de escritura*	*Comentarios*
Plantilla ROM	NMOS, CMOS	10–200 ns	4 semanas	Escribe una vez; baja potencia
Plantilla ROM	Bipolar	< 100 ns	4 semanas	Escribe una vez; alta potencia; baja densidad
PROM	Bipolar	< 100 ns	10–50 µs/byte	Escribe una vez; alta potencia; sin cargo por plantilla
EPROM	NMOS, CMOS	25–200 ns	10–50 µs/byte	Reutilizable; baja potencia; sin cargo por plantilla
EEPROM	NMOS	50–200 ns	10–50 µs/byte	límite de 10,000–100,000 escrituras/localidad

tenido deseado de la ROM, utilizando un disquete u otro medio de transferencia. El fabricante emplea esta información para crear una o más plantillas personalizadas para fabricar ROM con el patrón requerido. Los fabricantes de ROM imponen un *cargo por plantilla* de varios miles de dólares para los aspectos "personalizados" de la producción de plantillas ROM. Debido a los cargos por plantilla y al retraso de cuatro semanas que por lo común implica obtener los chips programados, las ROM de plantilla normalmente se utilizan en la actualidad sólo en aplicaciones de muy alto volumen. Para aplicaciones de bajo volumen existen opciones de costo más eficiente, que se discuten a continuación.

cargo por plantilla

memoria de sólo lectura programable

programador de PROM

enlace de fusible

Una *memoria de sólo lectura programable* (*PROM, programmable read-only memory*) es similar a una ROM de plantilla, excepto que el cliente puede almacenar valores de datos (por ejemplo, "programar la ROM") en sólo unos cuantos minutos con la ayuda de un *programador de PROM*. Un chip PROM es fabricado con todos sus diodos o transistores "conectados". Esto corresponde a tener todos los bits en un valor particular, casi siempre 1. El programador de PROM puede utilizarse para establecer los bits deseados al valor opuesto. En los PROM bipolares, esto se hace al vaporizar diminutos *enlaces de fusible* dentro del PROM correspondiente a cada bit. Un enlace se vaporiza al seleccionarlo utilizando las líneas de dirección y datos del PROM y aplicando luego un pulso de alto voltaje (10-30 V) al dispositivo a través de una terminal de entrada especial.

Los primeros PROM bipolares tenían problemas de confiabilidad. En ocasiones los bits almacenados cambiaban debido a los enlaces vaporizados (en forma incompleta) que se "regeneraban", y a veces ocurrían fallas intermitentes como consecuencia de los fragmentos que flotaban en el interior del encapsulado del CI. Sin embargo, estos problemas fueron resueltos y la tecnología de enlaces de fusibles confiable se usa hoy en día no solamente en PROM bipolares, sino también en los circuitos bipolares PLD que describimos en la sección 5.3.4.

memoria de sólo lectura programable borrable

Una *memoria de sólo lectura programable borrable* (*EPROM, erasable programmable read-only memory*) es programada como una PROM, pero también puede "borrarse" y regresar al estado de puros unos, si se le expone a la luz ultravioleta. ¡No, la luz no causa que los fusibles se regeneren! Más bien, los EPROM utilizan una tecnología diferente, llamada "MOS de compuerta flotante".

transistor MOS de compuerta flotante

Como se ilustra en la figura 10-10, un EPROM tiene un *transistor MOS de compuerta flotante* en cada localidad de bit. Cada transistor tiene dos compuertas. La compuerta "flotante" no está conectada y se encuentra rodeada de material de aislamiento con impedancia extremadamente alta. Para programar un EPROM, se aplica un voltaje alto a la compuerta no flotante para cada localidad de bit donde se vaya a almacenar un 0. Esto provoca una ruptura temporal en el material aislante y permite que se acumule una carga negativa en la compuerta flotante. Cuando se retira el voltaje alto, la carga negativa permanece. Durante operaciones de lectura subsiguientes, la carga negativa evita que el transistor MOS se encienda cuando es seleccionado.

Los fabricantes de EPROM "garantizan" que un bit programado en forma adecuada retendrá el 70% de su carga por al menos 10 años, incluso si la parte es almacenada a 125°C, de modo que el EPROM en definitiva pertenece a la categoría de la "memoria no volátil". Sin embargo, también se pueden borrar. El material aislante que rodea la compuerta flotante llega a ser ligeramente conductivo si se expone a luz ultravioleta con una cierta longitud de onda. De esta forma, los EPROM pueden borrarse al exponer los chips a la luz ultravioleta, por lo común durante un tiempo de 5 a 20 minutos. Un chip EPROM

borrado de EPROM

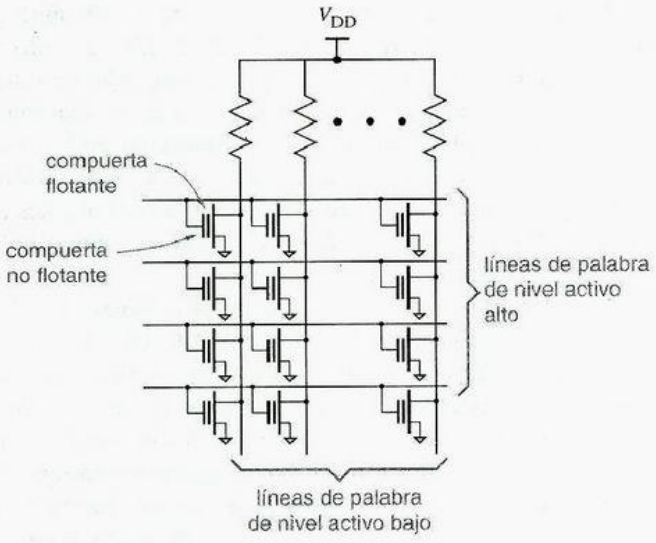


Figura 10-10
Matriz de almacenamiento en una EPROM utilizando transistores MOS de compuerta flotante.

se encuentra alojado normalmente en un encapsulado con una tapa de cuarzo a través de la cual el chip puede ser expuesto a la luz de borrado.

Es probable que la aplicación más común de las EPROM sea almacenar programas en sistemas microprocesadores. Las EPROM se utilizan por lo regular durante el desarrollo de los programas, cuando el programa u otra información en la EPROM debe modificarse varias veces durante la depuración. Sin embargo, las ROM y las PROM por lo regular cuestan menos que las EPROM de capacidad semejante. Por lo tanto, una vez que se finaliza un programa, puede utilizarse una ROM o una PROM en la producción para ahorrar costos. La mayor parte de las PROM actuales son en realidad EPROM alojadas en encapsulados económicos sin tapas de cuarzo; éstas se denominan *ROM programables una sola vez* (*OTP, one-time programmable*).

ROM programable una vez (OTP)

Una *memoria sólo de lectura programable eléctricamente borrable* (*EEPROM, electrically erasable programmable read-only memory*) es como una EPROM, excepto que los bits individuales que se almacenan se pueden borrar en forma eléctrica. Las compuertas flotantes en una EEPROM se encuentran rodeadas por una capa de aislamiento mucho más delgada y puede ser borrada mediante la aplicación de un voltaje de la polaridad opuesta a la del voltaje de carga para la compuerta no flotante. Las EEPROM grandes (1 Mbit o más grandes) permiten borrar sólo en bloques de tamaño fijo, de 128 a 512 Kbits (16-64 Kbytes) a un tiempo. Estas memorias se conocen por lo regular como *EPROM de destello (flash EPROM)* o *memorias "flash" o de destello*, porque el borrado ocurre "en un destello".

memoria de sólo lectura programable borrable eléctricamente

EPROM flash
memoria flash

Como se indicó en la tabla 10-5, la programación o "escritura" de una localidad EEPROM toma mucho más tiempo que leerla, de manera que un EEPROM no es sustituto para las memorias de lectura/escritura que se analizarán más adelante en este capítulo. También, debido a que la capa aislante es tan delgada, puede ser desgastada por operaciones repetidas de programación. Como resultado, las EEPROM pueden reprogramarse un número limitado de veces, hasta 10,000 veces por localidad. Por consiguiente, las EEPROM se emplean habitualmente para almacenar datos que deben ser conservados cuando el equipo no está energizado, pero que no cambian con mucha frecuencia tal como la configuración predeterminada para una computadora.

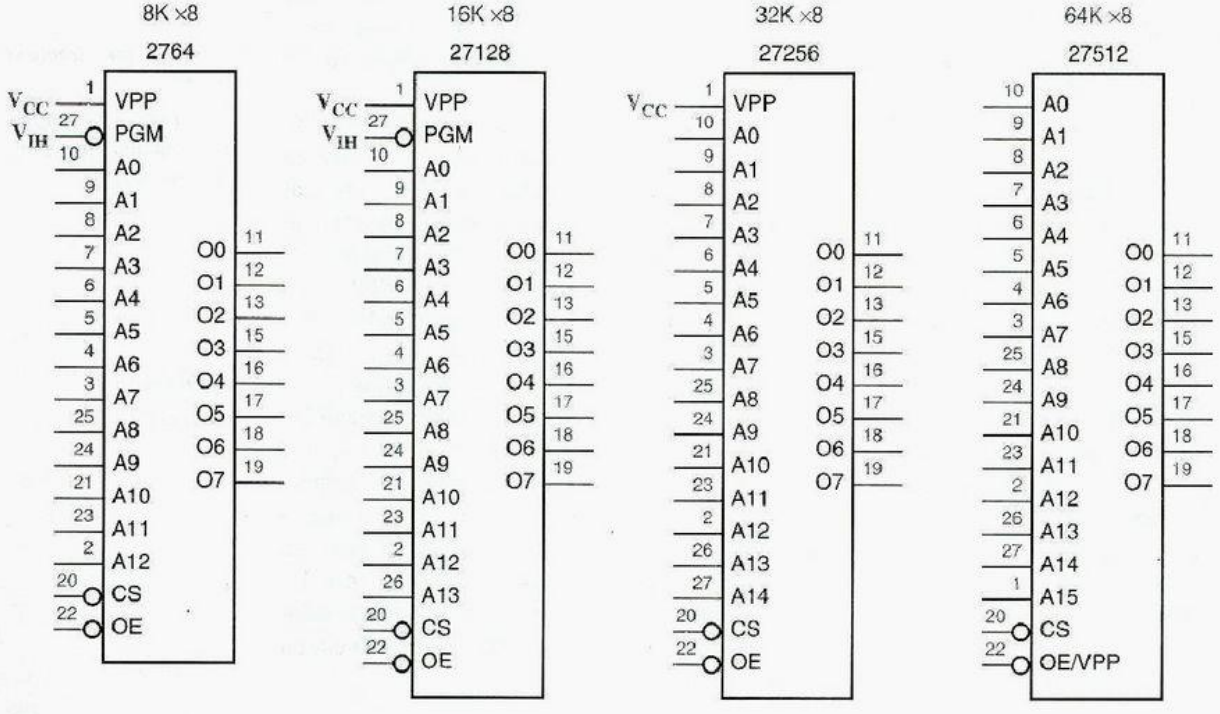
2764
27128
27256
27512

Las ROM populares para microprocesadores y otras aplicaciones ROM de velocidad moderada incluyen los EPROM *2764, 27128, 27256 y 27512*, cuyos símbolos lógicos se ilustran en la figura 10-11. La figura también indica las terminales que deben conectarse a una señal constante durante el funcionamiento normal sólo de lectura. Las entradas que están identificadas como V_{CC} deben conectarse a +5 volts, las entradas identificadas como V_{IH} deben conectarse a una señal lógica válida de nivel ALTO y "N.C." quiere decir "sin conexión" ("no connection"). Se emplean diferentes configuraciones para programar y probar el dispositivo; la terminal marcada como VPP se utiliza para aplicar el voltaje de programación.

Las ROM más grandes se encuentran disponibles con más bits y, en algunos casos, con salidas de datos más amplias de 16 o 32 bits. En 1999, uno de los más impresionantes era la memoria flash AMD 29LV640, con 64 Mbits en 256 sectores de 32 Kbytes. También se producen ROM mucho más pequeñas con interfaces serie de 3 bits para aplicaciones especializadas, como la descarga de la información de programación en FPGA.

Las memorias flash múltiples con frecuencia son encapsuladas en un módulo simple del tamaño de una tarjeta de crédito para aplicaciones que requieren de grandes cantidades de almacenamiento no volátil. La aplicación más común de estos módulos se encuentra en las cámaras digitales, donde el almacenamiento de una imagen de alta resolución puede requerir de aproximadamente 4 Mbites. En 1999, la tarjeta de memoria flash más grande fue desarrollada por el líder de la industria SanDisk Corporation y contenía 192 Mbytes (1536 Mbits) de memoria.

Figura 10-11 Símbolos lógicos para las EPROM estándar en encapsulados en doble línea de 28 terminales.



10.1.5 Temporización y entradas de control de ROM

Las salidas de una ROM a menudo deben ser conectadas a un bus de tres estados, donde diferentes dispositivos pueden controlar el bus en diferentes tiempos. Por tanto, la mayoría de los chips ROM comerciales tienen salidas de datos de tres estados y una *entrada de habilitación de salida (OE, output-enable)* que debe ser asertiva para habilitar las salidas.

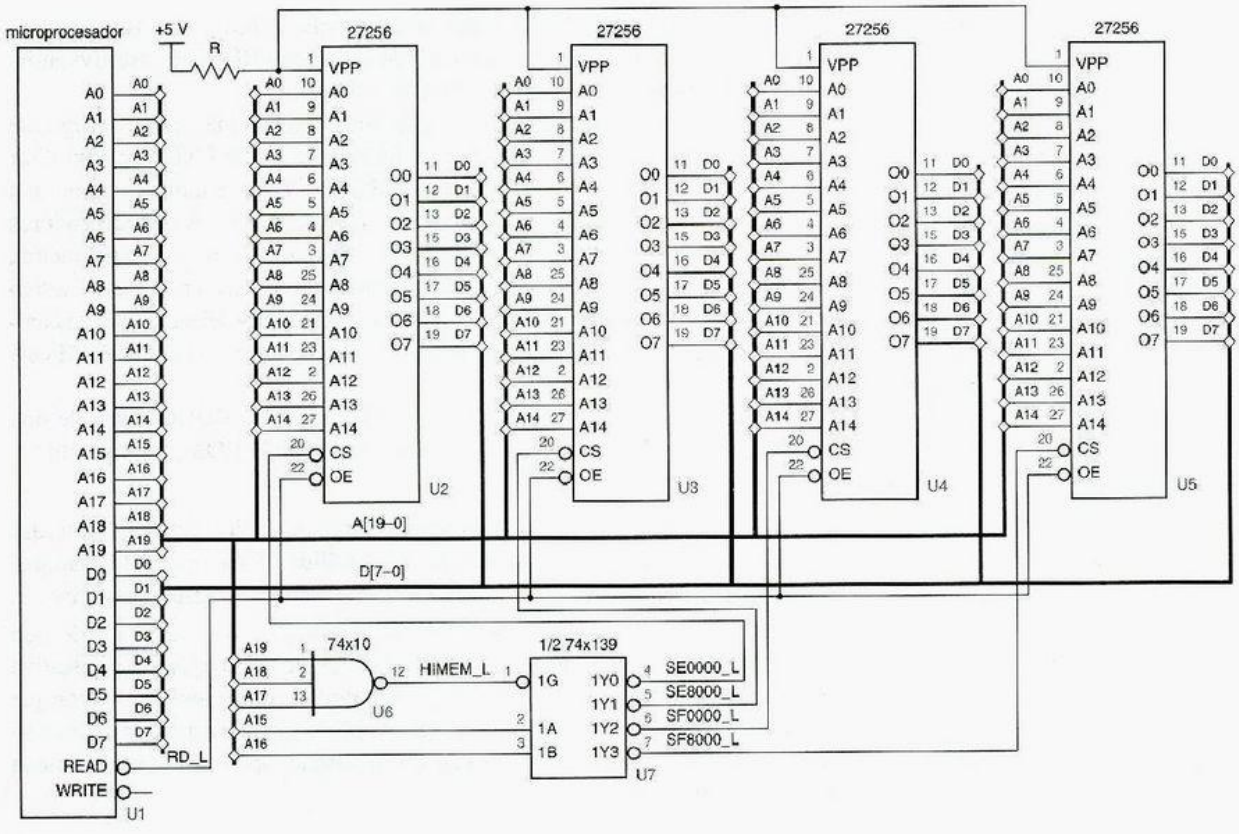
entrada de habilitación de salida (OE)

Muchas aplicaciones de las ROM, en especial el almacenamiento de programas, tienen múltiples ROM conectados a un bus, donde sólo una ROM controla el bus a la vez. La mayoría de las ROM tienen una *entrada de selección de chip (CS, chip-select)* para simplificar el diseño de tales sistemas. Además de la OE, una entrada CS de ROM debe activarse para habilitar las salidas de tres estados.

entrada de selección de chip (CS)

La figura 10-12 ilustra cómo podrían utilizarse las entradas OE y CS cuando se conectan cuatro ROM 32K × 8 a un sistema microprocesador que requiere 128 Kbytes de ROM. El microprocesador tiene un bus de datos de 8 bits y un bus de direcciones de 20 bits, para un máximo espacio de direccionamiento de 1 Mbyte (2^{20} bytes). Se supone que la ROM se encuentra localizada en los 128K más altos del espacio de direcciones. Para obtener este comportamiento, se emplea una compuerta AND para producir la señal HIMEM_L, la cual es asertiva cuando el bus de direcciones contiene una dirección en los 128K más altos (A19–A17 = 111). Un decodificador 74x139 selecciona entonces una de

Figura 10-12 Decodificación de dirección y habilitación de ROM en un sistema de microprocesador.



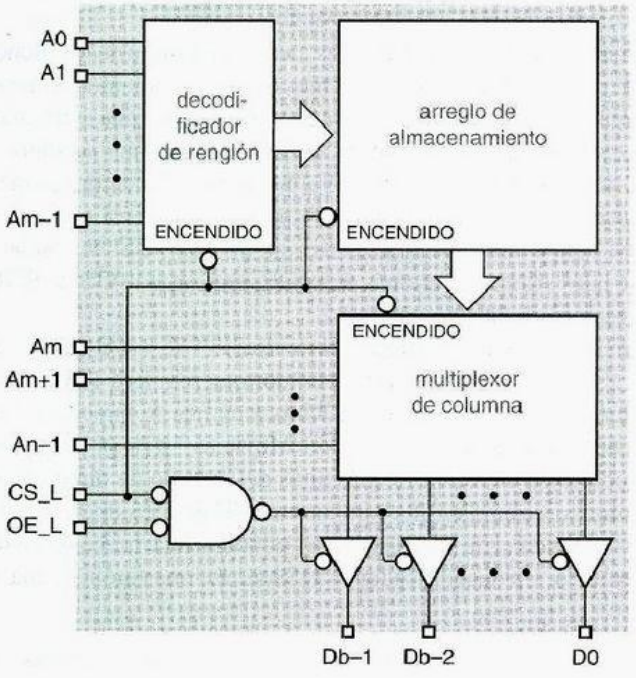


Figura 10-13
Estructura interna de la ROM, que indica las entradas de control.

las cuatro ROM de 32K × 8. La ROM de selección controla el bus de datos sólo cuando el microprocesador solicita una operación de lectura cuando READ es asertiva, misma que se encuentra conectada a todas las entradas OE.

Como lo hemos descrito hasta ahora, una entrada CS no es más que una segunda entrada de habilitación de salida, que es operada con OE a través de AND, para habilitar las salidas de tres estados. Sin embargo, en muchas ROM, CS sirve también como una *entrada de apagado*. Cuando se niega CS, la potencia se retira de los decodificadores internos de la ROM, controladores y multiplexores. En este *modo de espera* de operación, una ROM típica consume menos del 10% de la potencia que utiliza en el *modo activo* con CS asertiva. En la figura 10-12, al menos una ROM se encuentra seleccionada en cualquier momento, de modo que el consumo de potencia total de todos los chips ROM está mucho más cerca del de un chip que de cuatro.

entrada de apagado
modo de espera
modo activo

La figura 10-13 muestra cómo las entradas CS y OE se utilizan dentro de una ROM típica. La figura 10-14 ilustra la temporización típica de la ROM, incluyendo los parámetros siguientes:

t_{AA}
tiempo de acceso de dirección

t_{AA} *Tiempo de acceso de dirección.* Es el retardo de propagación desde las entradas de dirección estables hasta las salidas de datos válidas. Cuando los diseñadores hablan de "una ROM de 100 ns", por lo regular se refieren a este parámetro.

t_{ACS}
tiempo de acceso de selección de chip

t_{ACS} *Tiempo de acceso de selección de chip.* El tiempo de acceso de selección de chip de una ROM es el retardo de propagación desde el momento que CS es asertiva hasta que las salidas de datos son válidas. En algunos chips, esto es mayor que el tiempo de acceso de dirección, porque el chip toma un instante mientras se "calienta". En otros, este tiempo es más corto porque CS controla solamente la habilitación de salida.

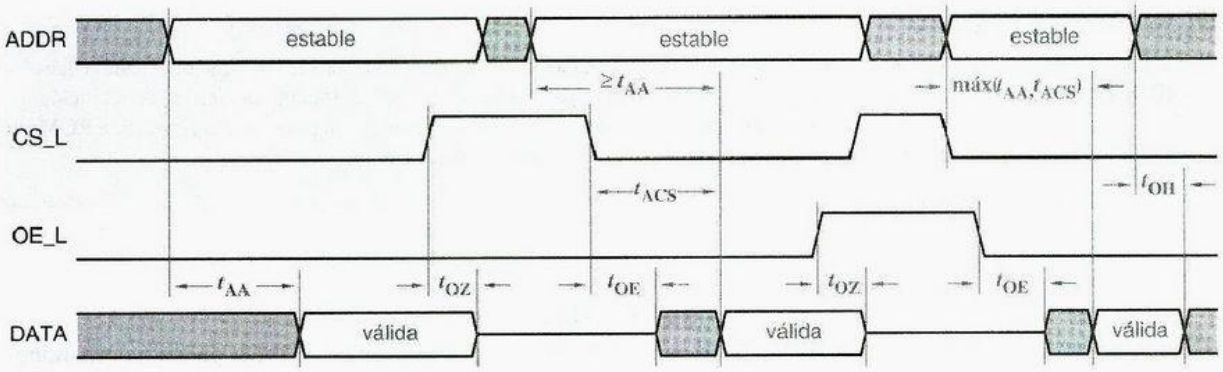


Figura 10-14 Temporización de ROM.

t_{OE} *Tiempo de habilitación de salida.* Este parámetro es en general mucho más corto que el tiempo de acceso. El tiempo de habilitación de salida de una ROM es el retardo de propagación desde OE y CS asertivas hasta que los controladores de salida de tres estados han dejado el estado Hi-Z. Dependiendo de si las entradas de dirección han estado estables el tiempo suficiente, los datos de salida pueden o no ser válidos en ese punto.

t_{OE}
tiempo de habilitación de salida

t_{OZ} *Tiempo de deshabilitación de salida.* El tiempo de deshabilitación de salida de una ROM es el retardo de propagación desde el momento en que OE o CS son negadas hasta que los controladores de salida de tres estados han entrado al estado Hi-Z.

t_{OZ}
tiempo de deshabilitación de salida

t_{OH} *Tiempo de retención de salida.* El tiempo de retención de salida de una ROM es el intervalo de tiempo en el que la salida permanece válida después de un cambio en las entradas de dirección, o después que OE_L o CS_L son negadas.

t_{OH}
tiempo de retención de salida

Como con otros componentes, el fabricante especifica valores máximos, y en ocasiones, típicos para todos los parámetros de temporización. Por lo regular, los valores mínimos también son especificados para t_{OE} y t_{OH}. El valor mínimo de t_{OH} se especifica habitualmente como 0; es decir, el retardo de lógica combinacional mínimo a través de la ROM es 0.

NO TODAS LAS ENTRADAS SE CREAN EN LA MISMA FORMA

De la estructura decodificadora y multiplexora de la figura 10-13, se puede pensar que el tiempo de acceso de dirección es más corto desde algunas entradas de dirección que de otras. En una aplicación dada, si algunas señales de entrada de dirección se retrasarán con relación a otras, se podría recuperar este retardo mediante la conexión de las señales más lentas a las entradas de ROM "más rápidas". Después de todo, cualquier señal de entrada podría conectarse a cualquier entrada de dirección de la memoria ROM, si usted estuviera dispuesto a reacomodar el contenido de la memoria ROM.

Sin embargo, los fabricantes de ROM no especifican cuáles entradas son más rápidas (si es que existen). De hecho, las características eléctricas internas de la mayoría de las memorias ROM minimizan la diferencia y por lo tanto no debemos preocuparnos por esto.

MEMORIAS ROM INCREÍBLEMENTE RÁPIDAS

El retardo real a través de una ROM nunca es 0, por supuesto, pero es fácil que sea apenas (ligeramente menos) de lo que se necesita para satisfacer un requerimiento de tiempo de retención distinto de cero en otra parte del diseño. De modo que es mejor suponer que t_{OH} para una ROM es 0, a menos que en realidad usted sepa lo que está haciendo.

10.1.6 Aplicaciones de ROM

Como ya lo mencionamos, la aplicación más común de las ROM es para almacenamiento de programa en sistemas microprocesadores. Sin embargo, en muchas aplicaciones una ROM puede proporcionar una realización de bajo costo de una función de lógica combinacional compleja o "aleatoria". En esta sección daremos un par de ejemplos de circuitos basados en ROM que se utilizan en la red telefónica digital.

Cuando una señal de voz analógica entra a un sistema telefónico digital típico, se muestrea 8,000 veces por segundo y se transforma en una secuencia de bytes de 8 bits que representan la señal analógica en cada punto de muestreo. En la sección 8.5.3 mostramos cómo estos muestreos de voz digital podían ser convertidos entre los formatos paralelo y serie, pero no describimos la codificación de los mismos bytes de 8 bits. Lo haremos ahora, y después mostraremos cómo los circuitos basados en ROM pueden tratar con facilidad esta información altamente codificada.

codificación lineal
rango dinámico

La codificación de 8 bits más simple del signo y la amplitud de una señal analógica sería un entero de 8 bits en el sistema de complemento a dos o de magnitud con signo; esto se conoce como una *codificación lineal*. Sin embargo, una codificación lineal de 8 bits produce un *rango dinámico* (la razón entre el intervalo de números representables y la diferencia más pequeña representable) de sólo 2^8 o 256. Para los audiófilos, esto corresponde al rango o intervalo dinámico en la potencia de la señal de 20 log 256 o aproximadamente 48 dB. En comparación, los discos compactos de audio utilizan una codificación lineal de 16 bits con un rango dinámico teórico de 20 log 2^{16} o cerca de 96 dB.

codificación de compresión-expansión
PCM de ley μ
PCM de ley A

En vez de una codificación lineal, la red telefónica norteamericana utiliza una *codificación de compresor-expansor* de 8 bits denominada *PCM (pulse code-modulation, modulación de pulsos codificados) de ley* μ. (La red europea utiliza una fórmula de compresión-expansión de 8 bits denominada *ley A*). La figura 10-15 ilustra el formato de un byte codificado de 8 bits, una especie de representación de punto flotante con campos de signo (S), exponente (E) y mantisa (M). El valor analógico V representado por un byte en este formato está dado por la fórmula

$$V = (1-2S) \cdot [(2^E) \cdot (2M + 33) - 33]$$

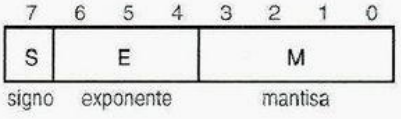


Figura 10-15
Formato de un byte PCM de ley μ.

DINERO POR NADA... Con la PCM de ley μ no se obtiene algo por nada, por supuesto. En la codificación de compresión-expansión, la diferencia entre valores codificados sucesivos es más grande a amplitudes altas que a bajas amplitudes, de modo que las codificaciones de señales analógicas grandes sufren de mayor distorsión cuantizada (provocada por la diferencia entre el valor analógico muestreado y la representación codificada disponible más cercana). Expresada como un porcentaje de la amplitud de señal pico, sin embargo, la distorsión cuantizada es casi constante a través de todo el intervalo de valores representables.

Una señal analógica representada en este formato puede abarcar desde $-8159 \cdot k$ hasta $+8159 \cdot k$, donde k es un factor de escala arbitrario. El intervalo de señales es $2 \cdot 8159$ y la diferencia más pequeña que puede ser representada es sólo 2 (cuando $E = 0$), de modo que el rango dinámico es 20 log 8159 o aproximadamente 78 dB, una mejora bastante apreciable sobre el código lineal de 8 bits.

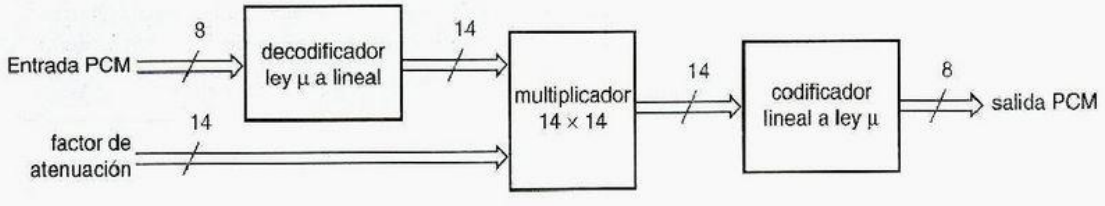
Ahora examinemos algunas aplicaciones de ROM que involucran señales de voz codificadas en ley μ. Aunque usted no lo crea, en muchos tipos de conexiones telefónicas su voz es atenuada a propósito en unos cuantos decibelios para hacer que las cosas trabajen mejor. En una red telefónica analógica, la atenuación se realiza mediante un simple circuito analógico pasivo, pero no es así en el mundo digital. Dado un byte PCM de ley μ, un *atenuador digital* debe producir un byte PCM diferente que represente la señal analógica original multiplicada por un factor de atenuación especificado.

atenuador digital

Una manera de construir un atenuador digital se muestra en la figura 10-16. El byte de entrada es aplicado a un decodificador de ley μ que expande el byte de acuerdo con la fórmula dada antes para producir un entero con signo de magnitud de 14 bits. Este valor lineal de 14 bits se multiplica entonces por una fracción binaria de 14 bits correspondiente a la cantidad de atenuación deseada. Los bits fraccionales del producto se descartan y el resultado es codifica en un nuevo byte PCM de ley μ de 8 bits. Cada bloque en la figura podría ser construido quizás con una docena de chips MSI, un CPLD o un FPGA.

La figura 10-17 es el diagrama lógico para un atenuador digital que emplea una sola ROM económica de 8K×8 en su lugar. Esta ROM puede aplicar cualquiera de 32 diferentes factores de atenuación a un byte de entrada de ley μ: simplemente almacena 32 tablas diferentes de atenuación. Los bits de dirección de orden superior seleccionan

Figura 10-16 Diagrama de bloques de un atenuador digital utilizando componentes discretos.



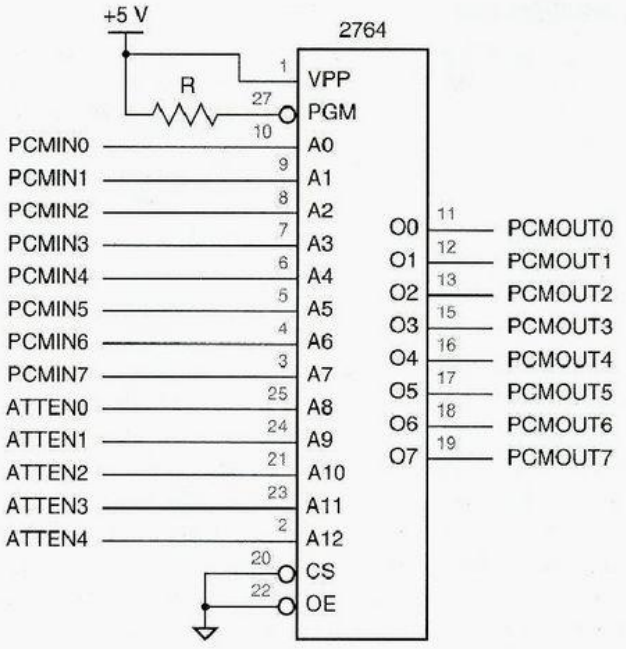


Figura 10-17
Un atenuador digital.

una tabla, y los bits de dirección de orden inferior seleccionan una entrada. Cada entrada es un byte de ley μ que corresponde al código de atenuación dado y el byte de entrada. La tabla 10-6 es un programa en C que genera el contenido de la ROM. Los detalles de las funciones `UlawToLinear` y `LinearToUlaw` se dejan como un ejercicio.

Otra aplicación de la ROM en la telefonía digital es un circuito de conferencia digital. En la red telefónica analógica, es bastante fácil hacer una conexión de conferencia entre tres o más partes. Sólo conecte los alambres del teléfono analógico y se obtiene una unión de agregación analógica, de modo que cada persona escucha a todos los demás (no es tan fácil hacerlo bien, pero se ha entendido la idea). En la red digital, por supuesto, resultaría un caos si sólo se pusieran en corto las señales de salida digitales. En cambio, un circuito de conferencia digital debe incluir un *sumador* digital que produce muestras de salida correspondientes a sumas de las muestras de entrada.

...Y LA VERIFICACIÓN ES GRATIS

El atenuador digital es otro buen ejemplo de los muchos intercambios de eficiencia en costos entre espacio y tiempo que llegaron a ser posibles cuando la red telefónica "se hizo digital". Una sola señal de voz analógica produce una muestra PCM de ley μ de 8 bits una vez cada 125 μs, pero la ROM del atenuador digital puede producir una salida válida en unos cuantos cientos de nanosegundos o menos. De esta forma, en un sistema telefónico digital, un solo chip de ROM puede atenuar cientos de flujos de voz digital, donde cientos de redes atenuadoras analógicas se habrían requerido en los viejos tiempos.

Tabla 10-6 Programa para generar el contenido de un ROM atenuador de 8K × 8, de 32 posiciones para bytes codificados de ley μ.

```
#include <stdio.h>
#include <math.h>

extern int UlawToLinear(int in);
extern int LinearToUlaw(int x);

void main()
{
  int i, j, position;
  int pcmIN, linearOUT, pcmOUT;
  double atten, attenDB, fpcmOUT;

  for (position=0; position<=31; position++) {  /* Make 32 256-byte tables.*/
    printf("%i attenuation (dB): ");  /* Get amount in dB from designer,          */
    scanf("%f\n", attenDB);       /* negative for attenuation, positive for again. */
    atten = exp(log(10)*attenDB/10); /* Convert to fraction */
    for (i=0; i<=15; i++) {  /* Construct output file in rows of 16. */
      printf("%04x:", position*256 + i*16);
      for (j=0; j<=15; j++) {
        pcmIN = i*16 + j;
        fpcmOUT = atten * UlawToLinear(pcmIN);
        if (fpcmOUT >=0) linearOUT = floor(fpcmOUT + 0.5);  /* Redondeo */
        else linearOUT = ceil(fpcmOUT - 0.5);
        pcmOUT = LinearToUlaw(linearOUT);
        printf(" %2x", pcmOUT);
      }
      printf("\n");
    }
  }
}
```

Usted sabe cómo construir un sumador binario para dos operandos de 8 bits, pero los sumadores binarios no pueden procesar directamente los bytes PCM de ley μ. En su lugar, los bytes PCM de 8 bits deben convertirse en el formato lineal de 14 bits, luego sumarse, y posteriormente recodificarse. Como en el circuito atenuador digital de la figura 10-16, un diseño MSI para esta función requeriría muchos chips. Alternativamente, la función podría realizarse mediante una simple ROM de 64K × 8, como se ilustra en la figura 10-18. La ROM tiene 16 entradas de dirección, acomodando dos operandos PCM de ley μ de 8 bits. Para cada par de valores de operandos, la dirección de ROM correspondiente contiene la suma PCM de ley μ previamente calculada. La tabla 10-7 es un programa en C que puede ser utilizado para generar el contenido de la ROM.

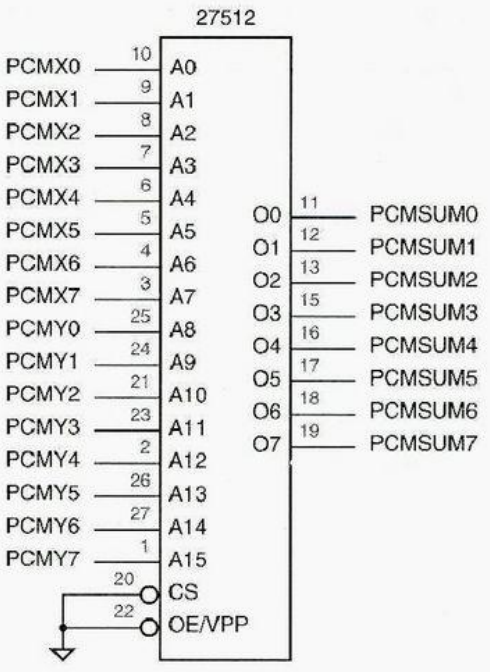


Figura 10-18
Un circuito sumador para bytes codificados de ley μ.

Estos ejemplos ilustran las muchas ventajas de construir funciones combinacionales complejas con las ROM. Por lo regular, consideramos que una función es "compleja" si sabemos que la construcción de un circuito al nivel de compuerta para la función será un verdadero dolor. Sin embargo, la mayoría de las funciones "complejas", como los ejemplos en esta subsección, tienen descripciones en palabras bastante directas. Una descripción así en general puede ser traducida en un programa de computadora que "deduzca" lo que deberían ser las salidas de la función para cada posible combinación de entrada, de modo que la función pueda ser construida tan sólo poniendo su tabla de verdad en una ROM.

Además de la facilidad y velocidad de diseño, un circuito basado en ROM tiene otras importantes ventajas:

- Para una función moderadamente compleja, un circuito basado en ROM es por lo regular más rápido que un circuito que utilice dispositivos SSI/MSI múltiples y PLD, y a menudo más rápido que un FPGA o chip LSI personalizado en una tecnología comparable.
- El programa que genera el contenido de la ROM puede ser estructurado con facilidad para manejar casos inusitados o indefinidos que requerirían de hardware adicional en cualquier otro diseño. Por ejemplo, el programa sumador de la tabla 10-7 maneja sin dificultad sumas fuera del intervalo. (También véase el ejercicio 10.22.)
- Una función de ROM es fácil de modificar tan sólo cambiando el patrón almacenado, habitualmente sin modificar ninguna conexión externa. Por ejemplo, el atenuador PCM y las ROM de sumador en esta subsección pueden cambiarse para usar PCM de ley A de 8 bits, la codificación de voz digital estándar en Europa.

Tabla 10-7 Programa para generar el contenido de una ROM sumadora de 64K × 8 para bytes codificados de ley μ.

```
#include <stdio.h>
#include <math.h>
#define MINLINEAR -8159
#define MAXLINEAR 8159

void main()
{
  int i, j, linearSum;
  int pcmINX, pcmINY;

  for (pcmINY=0; pcmINY<=255; pcmINY++) {  /* For all Y samples... */
    for (i=0; i<=15; i++) {  /* Construct output file in rows of 16. */
      printf("%04x:", pcmINY*256 + i*16);
      for (j=0; j<=15; j++) {  /* For all X samples... */
        pcmINX = i*16 + j;
        linearSum = UlawToLinear(pcmINX) + UlawToLinear(pcmINY);
        /* The next two lines perform "clipping" on overflow. */
        if (linearSum < MINLINEAR) linearSum = MINLINEAR;
        if (linearSum > MAXLINEAR) linearSum = MAXLINEAR;
        printf(" %02x", LinearToUlaw(linearSum));
      }
      printf("\n");
    }
  }
}
```

- Los precios de la ROM y otros dispositivos lógicos estructurados siempre se encuentran disminuyendo, haciéndolos más económicos, y sus densidades siempre se encuentran aumentando, expandiendo el ámbito de los problemas que pueden resolverse con un solo chip.

Existen también unas cuantas desventajas de los circuitos basados en ROM:

- Para funciones desde simples hasta moderadamente complejas, un circuito basado en ROM puede costar más, consumir más potencia o funcionar con más lentitud que un circuito que utiliza unos cuantos dispositivos SSI/MSI y PLD o un pequeño FPGA.
- Para funciones con más de 20 entradas, un circuito basado en ROM es poco práctico debido al límite de tamaños de ROM que están disponibles. Por ejemplo, usted no querría construir un sumador de 16 bits en ROM: requeriría miles y miles de millones de bits.

10.2 Memoria de lectura/escritura

memoria de lectura/escritura (RWM)

memoria de acceso aleatorio (RAM)

El nombre *memoria de lectura/escritura (RWM, read/write memory)* se da a los arreglos de memoria en los cuales podemos almacenar y recuperar información en cualquier momento. La mayoría de las RWM utilizadas en sistemas digitales en la actualidad son *memorias de acceso aleatorio (RAM, random-access memory)*, lo que significa que el tiempo que les toma leer o escribir un bit de memoria es independiente de la localidad del bit en la RAM. Desde este punto de vista, las ROM son también memorias de acceso aleatorio, pero el nombre "RAM" en general se emplea sólo para las memorias de acceso aleatorio de lectura/escritura.

RAM estática (SRAM)

RAM dinámica (DRAM)

En una *RAM estática (SRAM* o "S-ram"), se escribe una palabra una vez en una localidad, permanece almacenada mientras se aplica la potencia al chip, a menos que la misma localidad sea escrita otra vez. En una *RAM dinámica (DRAM* o "D-ram"), los datos almacenados en cada localidad deben refrescarse en forma periódica al leerlos y luego escribirlos otra vez, o si no desaparecen. Analizaremos ambos tipos en esta sección.

memoria volátil

memoria no volátil

RAM ferroeléctrica

La mayoría de las RAM pierden su memoria cuando se interrumpe el suministro de energía; son una forma de *memoria volátil*. Algunas RAM retienen su memoria incluso cuando el suministro de energía se interrumpe; se conocen como *memoria no volátil*. Ejemplos de RAM no volátiles son las antiguas memorias de núcleo magnético y las modernas memorias estáticas CMOS en un encapsulado extra grande que incluye una batería de litio con un tiempo de vida de 10 años. Hace poco se introdujeron las *RAM ferroeléctricas* no volátiles; estos dispositivos combinan elementos magnéticos y eléctronicos en un solo chip de CI que retiene su estado incluso cuando no se suministra energía, del mismo modo que las antiguas memorias de núcleo.

10.3 RAM estática

10.3.1 Entradas y salidas de la RAM estática

Al igual que una ROM, una RAM tiene entradas de dirección y control y salidas de datos, pero también tiene entradas de datos. Las entradas y salidas de una simple RAM estática de $2^n \times b$ bits se muestran en la figura 10-19. Las entradas de control son comparables a las de una ROM, con el agregado de una *entrada de habilitación de escritura (WE, write-enable)*. Cuando WE es asertiva, las entradas de datos se escriben en la localidad de memoria seleccionada.

entrada de habilitación de escritura (WE)

Las localidades de memoria en una RAM estática se comportan como latchs D, más que como flip-flops D disparados por flanco. Esto significa que cada vez que la entrada WE es afirmada, los latchs para la localidad de memoria seleccionada están "abiertos" (o "transparentes") y los datos de entrada fluyen dentro y a través del latch. El valor real almacenado es cualquiera que esté presente cuando se cierra el latch.

La RAM estática por lo normal tiene sólo dos operaciones de acceso definidas:

Lectura Una dirección se coloca en las entradas de dirección mientras que CS y OE son asertivas. Las salidas del latch para la localidad de memoria seleccionada se liberan hacia DOUT.

Escritura Una dirección se coloca en las entradas de dirección y una palabra de datos se coloca en DIN; entonces CS y WE son asertivas. Los latchs en la localidad de memoria seleccionada se abren, y la palabra de entrada se almacena.

MEMORIA DE ACCESO SERIE

La memoria de acceso aleatorio puede ser contrastada con la *memoria de acceso serie*, donde alguna localidad se encuentra inmediatamente accesible en cualquier momento, pero otras localidades se llevan pasos adicionales para tener acceso a ellas.

Algunas computadoras antiguas empleaban dispositivos electromecánicos de memoria de acceso serie, tales como líneas de retardo y tambores giratorios. Las instrucciones y los datos eran almacenados en un medio giratorio con sólo una localidad debajo de la "cabeza de lectura/escritura" en cualquier instante. Para tener acceso a una localidad aleatoria, la máquina tenía que esperar hasta que la rotación constante transportara esa localidad hasta situarla debajo de la cabeza.

En la década de 1970, fueron desarrollados los equivalentes electrónicos de las memorias de rotación de acceso serie, incluyendo memorias basadas en dispositivos de carga acoplada (CCD, charge-coupled devices) y otros que empleaban burbujas magnéticas. Ambos tipos de dispositivos eran aproximadamente equivalentes a registros de corrimiento de entrada y salida serie muy grandes con su salida serie conectada de regreso a la entrada serie. Este punto de conexión era el equivalente lógico de una "cabeza de lectura/escritura" de un disco duro. Para leer una localidad en particular, se cronometraba el registro de corrimiento hasta que el bit deseado aparecía en la salida serie, y para escribir en esa localidad, se sustituía el nuevo valor deseado en la entrada serie.

Aunque ofrecían una densidad mayor (más bits) que las DRAM en el momento en que fueron desarrolladas, las memorias CCD y de burbuja magnética nunca ganaron mucha aceptación comercial. Una razón de esto era la enorme inconveniencia del acceso serie. Otra era que nunca estuvieron más de un par de años por delante de las DRAM en cuanto a las densidades que podían conseguir.

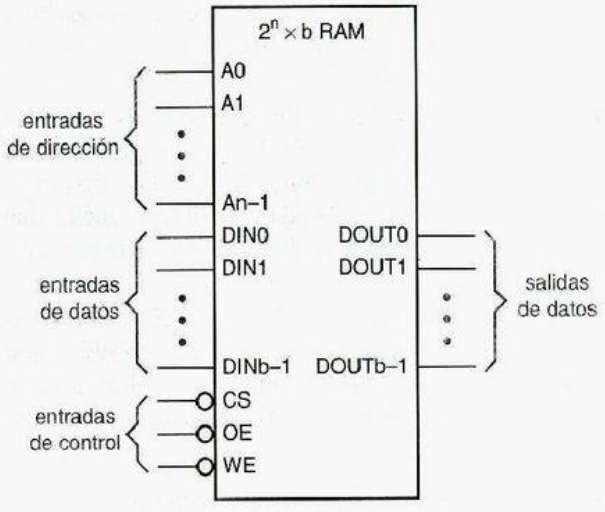


Figura 10-19
Estructura básica de una RAM de $2^n \times b$.

Es necesaria cierta cantidad de precaución cuando se tiene acceso a SRAM, porque es posible de manera inadvertida dañar a una o más localidades mientras se escribe en una seleccionada, si los requerimientos de temporización de la SRAM no se satisfacen. La subsección siguiente proporciona detalles acerca de la estructura interna de la SRAM para mostrar por qué sucede esto, y la que le sigue explica los requerimientos y comportamiento de temporización real.

10.3.2 Estructura interna de la RAM estática

celda de SRAM

Cada bit de memoria (o *celda de SRAM*) en una RAM estática tiene el mismo comportamiento funcional que el circuito en la figura 10-20. El dispositivo de almacenamiento en cada celda es un latch D. Cuando una entrada SEL_L de la celda es asertiva, los datos almacenados se colocan en la salida de la celda, que está conectada a una línea de bit. Cuando tanto SEL_L como WR_L son asertivas, el latch se abre y se almacena un nuevo bit de datos.

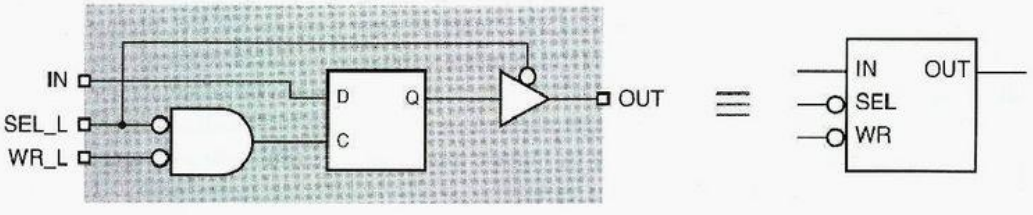
Las celdas SRAM son combinadas en un arreglo con lógica de control adicional para formar una RAM por completo estática, como se muestra en la figura 10-21 para una SRAM de 8×4. Como en una ROM simple, un decodificador en las líneas de dirección selecciona un renglón particular de la SRAM para que se tenga acceso en cualquier momento.

Aunque la figura 10-21 es un modelo un tanto simplificado de la estructura interna de la SRAM, retrata con precisión varios aspectos importantes del comportamiento de la SRAM:

- Durante operaciones de lectura, la salida de datos es una función combinacional de las entradas de dirección, como en una ROM. Ningún daño se produce al cambiar las líneas de dirección mientras el bus de datos de salida se encuentra habilitado. El tiempo de acceso para las operaciones de lectura está especificado desde el tiempo en que la última entrada de dirección se hace estable.
- Durante las operaciones de escritura, los datos de entrada se almacenan en latchs. Esto significa que los datos deben satisfacer ciertos tiempos de establecimiento y retención con respecto al flanco *posterior* de la señal de habilitación del latch. Es decir, los datos de entrada para una entrada D del latch, no necesitan ser estables en el momento en que WR_L es afirmada en forma interna; sólo deben ser estables cierto tiempo antes de que WR_L sea negada.
- Durante las operaciones de escritura, las entradas de dirección *deben* ser estables para un cierto tiempo de establecimiento antes de que WR_L sea afirmada de manera interna y para un tiempo de retención después que se niega WR_L. De otro modo, los datos se pueden "dispersar" sobre el arreglo debido a las interferencias que pueden aparecer en las líneas SEL_L cuando las entradas de dirección del decodificador son cambiadas.
- Internamente, WR_L es asertiva sólo cuando tanto CS_L como WE_L son asertivas. Por tanto, un *ciclo de escritura* comienza cuando tanto CS_L como WE_L son asertivas, y finaliza cuando cualquiera es negada. Los tiempos de establecimiento y retención para direcciones y datos son especificados con respecto a estos eventos.

ciclo de escritura

Figura 10-20 Comportamiento funcional de una celda de RAM estática.



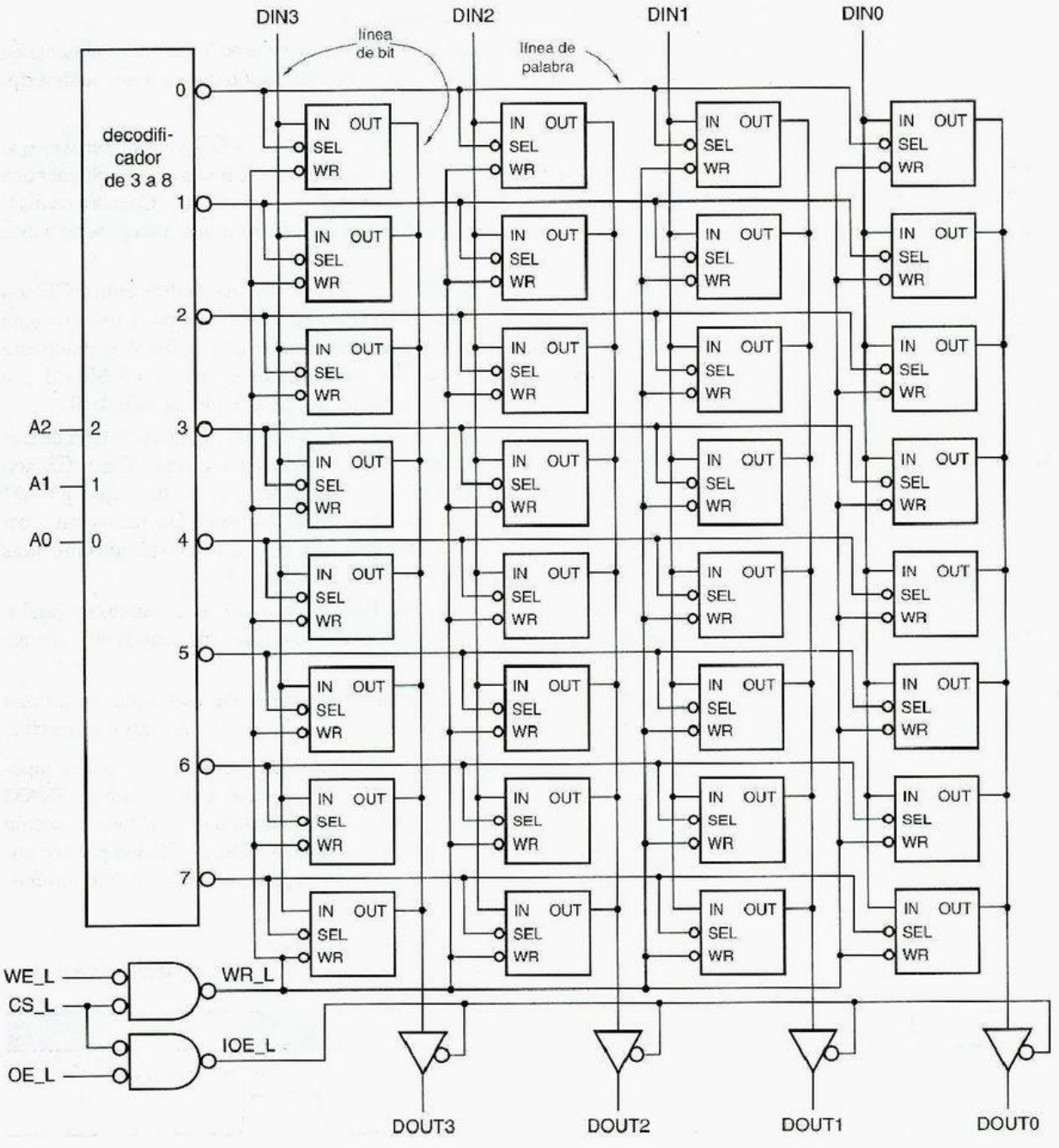


Figura 10-21 Estructura interna de una RAM estática de 8×4.

10.3.3 Temporización de RAM estática

La figura 10-22 muestra los parámetros de tiempo que se especifican normalmente en las operaciones de lectura en una RAM estática; a continuación presentamos su descripción breve:

t_{AA}
tiempo de acceso de dirección

t_{AA} *Tiempo de acceso de dirección.* Pensemos que OE y CS ya son asertivas o si pronto lo serán (no importa) esto es, cuánto tiempo se necesita para obtener una salida estable de datos después de un cambio en la dirección. Cuando los diseñadores hablan acerca de una "SRAM de 70 ns", se refieren en general a este número.

t_{ACS}
tiempo de acceso de selección de chip

t_{ACS} *Tiempo de acceso de selección de chip.* Suponemos que la dirección y OE son estables, o pronto lo serán (no importa) esto es, cuánto tiempo se necesita para obtener una salida estable de datos después que CS es asertiva. A menudo este parámetro es idéntico a t_{AA}. Pero en ocasiones es mayor en las SRAM que tienen modo de "apagado", y más corto en las SRAM que carecen de él.

t_{OE}
tiempo de habilitación de salida

t_{OE} *Tiempo de habilitación de salida.* Es el tiempo que necesitan los buffers de tres estados para abandonar el estado de alta impedancia, cuando OE y CS son asertivas. En general este parámetro es menor que t_{ACS}, de modo que la RAM puede tener acceso a los datos en forma interna antes que OE sea asertiva; esta característica se emplea para conseguir tiempos de acceso rápidos mientras que se evita un "conflicto de bus" en muchas aplicaciones.

t_{OZ}
tiempo de deshabilitación de salida

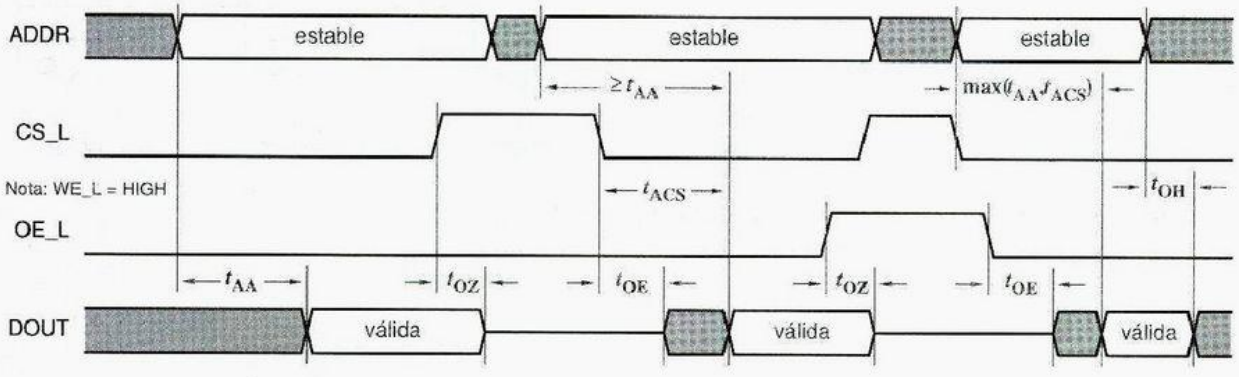
t_{OZ} *Tiempo de deshabilitación de salida.* Es el tiempo que necesitan los separadores de salida de tres estados para entrar al estado de alta impedancia después que se han negado OE_L o CS_L.

t_{OH}
tiempo de retención de salida

t_{OH} *Tiempo de retención de salida.* Este parámetro especifica cuánto tiempo los datos de salida permanecen válidos después de un cambio en las entradas de dirección.

Si usted ha estado poniendo atención, puede haber notado que el diagrama de temporización y los parámetros de temporización para las operaciones de lectura de SRAM son idénticas a lo que discutimos para las operaciones de lectura de ROM en la sección 10.1.5. Así son las cosas, cuando no se está escribiendo en ellas, las SRAM pueden utilizarse justo como las ROM. Esta afirmación no es cierta para las DRAM, como veremos más tarde.

Figura 10-22 Parámetros de temporización para operaciones de lectura en una RAM estática.



Los parámetros de tiempo para operaciones de escritura se muestran en la figura 10-23 y se describen a continuación.

t_{AS} *Tiempo de establecimiento de dirección antes de la escritura.* Todas las entradas de dirección deben ser estables en este tiempo, antes que CS y WE sean asertivas. De lo contrario se pueden corromper los datos que se almacenan en las localidades imprevisibles.

t_{AS}
tiempo de establecimiento de dirección

t_{AH} *Tiempo de retención de dirección después de escritura.* Análogo a t_{AS}, todas las entradas de dirección deben mantenerse estables hasta este tiempo, después que CS o WE sean negadas.

t_{AH}
tiempo de retención de dirección

t_{CSW} *Establecimiento de selección de chip antes del fin de la escritura.* CS debe ser asertiva por lo menos este tiempo antes del final del ciclo de escritura a fin de seleccionar una celda.

t_{CSW}
establecimiento de selección de chip

t_{WP} *Anchura del pulso de escritura.* WE debe ser asertiva por lo menos este tiempo para asegurar de manera confiable los datos que están en la celda seleccionada.

t_{WP}
anchura de pulso de escritura

t_{DS} *Tiempo de establecimiento de datos antes del fin de la escritura.* Todas las entradas de datos deben ser estables en este tiempo antes que finalice el ciclo de escritura. De otro modo, no se asegura la integridad de los datos.

t_{DS}
tiempo de establecimiento de datos

t_{DH} *Tiempo de retención de datos después del fin de escritura.* Análogo a t_{DS}, todas las entradas de datos deben mantenerse estables hasta este tiempo, después que termina el ciclo de escritura.

t_{DH}
tiempo de retención de datos

Los fabricantes de SRAM especifican dos tipos de ciclo de escritura, *controlados por WE* y *CS*, como se muestra en la figura. La única diferencia entre estos ciclos es si WE o CS es el último en ser asertivo y el primero en ser negado cuando se habilita la operación de escritura interna de la SRAM.

escritura controlada por WE

escritura controlada por CS

Los requerimientos de temporización de escritura de las SRAM podrían relajarse un poco si, en lugar de utilizar latchs, las celdas incluyeran flip-flops tipo-D disparados por flanco, con una entrada de reloj común y entradas de habilitación que están unidas y controladas por SEL y WR. Sin embargo, esto no es posible porque se duplicaría el área del chip de cada celda, ya que un flip-flop D está formado por dos latchs. De este modo, el diseñador lógico se deja para reconciliar la temporización tipo latch de la SRAM con el registro disparado por flanco y la temporización de máquina de estado utilizada en otra parte en un sistema.

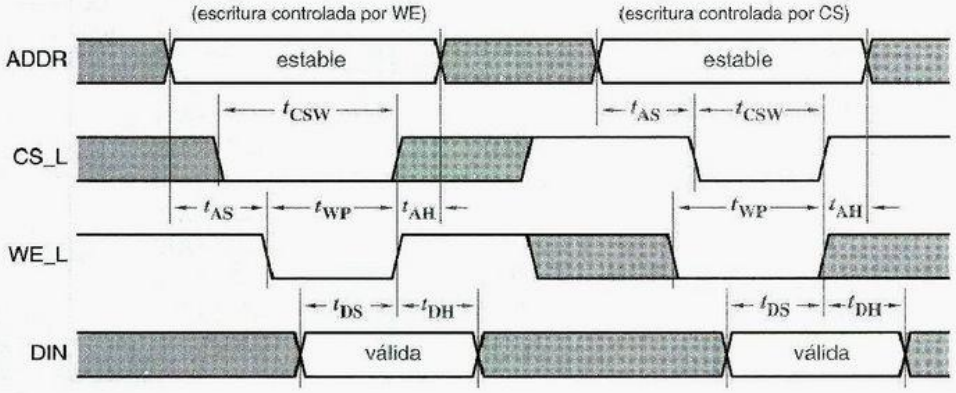


Figura 10-23
Parámetros de temporización para operaciones de escritura en una SRAM estática.

Una SRAM grande no contiene un arreglo físico cuyas dimensiones igualan las dimensiones lógicas de la memoria. Como en una ROM, las celdas de la SRAM se presentan en un arreglo casi cuadrado, y un renglón entero se lee de manera interna durante las operaciones de lectura. Por ejemplo, el esquema de un chip de SRAM de 32K × 8 puede ser muy semejante al de una ROM de 32K × 8 mostrada en la figura 10-9 de la página 841. Durante las operaciones de lectura, los multiplexores de columna pasan los bits de datos requeridos al bus de datos de salida, como se especifica por medio de un subconjunto de los bits de dirección (A5-A0 en el ejemplo de la ROM). Para operaciones de escritura, la circuitería de habilitación de escritura está diseñada de modo que sólo una columna en cada subarreglo se encuentre habilitada, como se determinan mediante el mismo subconjunto de los bits de dirección.

10.3.4 RAM estáticas estándar

Las RAM estáticas se encuentran disponibles en muchos tamaños y velocidades. En 1999, las SRAM convencionales más grandes utilizaban tecnología CMOS, estaban organizadas como de 256K × 16, 512K × 8, o 1M × 4 bits (4 Mbits), y tenían tiempos de acceso tan rápidos como de 10 ns. Las SRAM más rápidas compatibles con TTL/CMOS también empleaban tecnología CMOS y contenían nada más 1K × 4 bits (4 Kbits) ¡pero tenían tiempos de acceso de 2.7 ns!

Los números de partes para componentes SRAM no están estandarizados entre los fabricantes, aunque los componentes mismos con frecuencia tienen terminales de salida

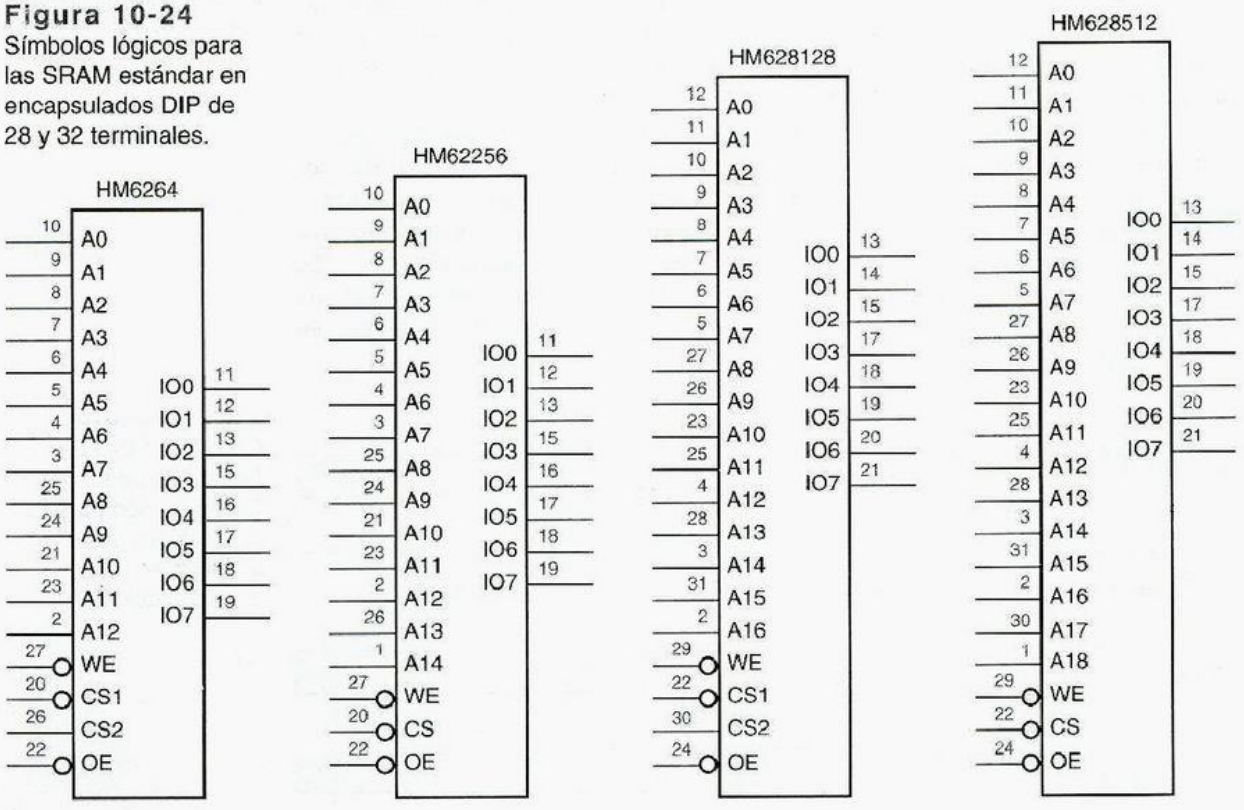


Figura 10-24
Símbolos lógicos para las SRAM estándar en encapsulados DIP de 28 y 32 terminales.

idénticas y son intercambiables. La figura 10-24 muestra los números de partes y terminales de salida para las memorias SRAM Hitachi, que abarcan en dimensiones desde las 8K × 8 estándar hasta las de 512K × 8, que usted puede utilizar en un proyecto de laboratorio de diseño digital; todas ellas emplean una fuente de alimentación de 5 V y vienen en encapsulados DIP. Estos dispositivos también se encuentran disponibles en encapsulados más compactos y los dispositivos más novedosos SRAM por lo regular se encuentran disponibles sólo en encapsulados compactos y utilizan voltajes de alimentación más bajos, como de 3.3 V o 2.5 V.

Las primeras dos SRAM de la figura utilizan encapsulados DIP de 28 terminales y tienen terminales de salida que están diseñadas como las de los EPROM de la figura 10-11. La SRAM *HM6264* de 8K × 8 es similar a un EPROM 2764, excepto porque tiene una entrada de habilitación de escritura y dos entradas de selección de chip en lugar de una; CS1_L y CS2 deben ser asertivas en conjunto para habilitar el chip. La SRAM *HM62256* de 32K × 8 es semejante a una EPROM 27256, con la adición de una entrada de habilitación de escritura. Ambos chips se encuentran disponibles con muchos fabricantes, con frecuencia con diferentes números de parte, y tienen tiempos de acceso en el intervalo de los 15-150 ns. Las dos últimas SRAM en la figura utilizan encapsulados DIP de 32 terminales. El *HM628128* es una SRAM de 128K × 8 y el *HM628512* es una SRAM de 512K × 8.

HM6264

HM62256

HM628128

HM628512

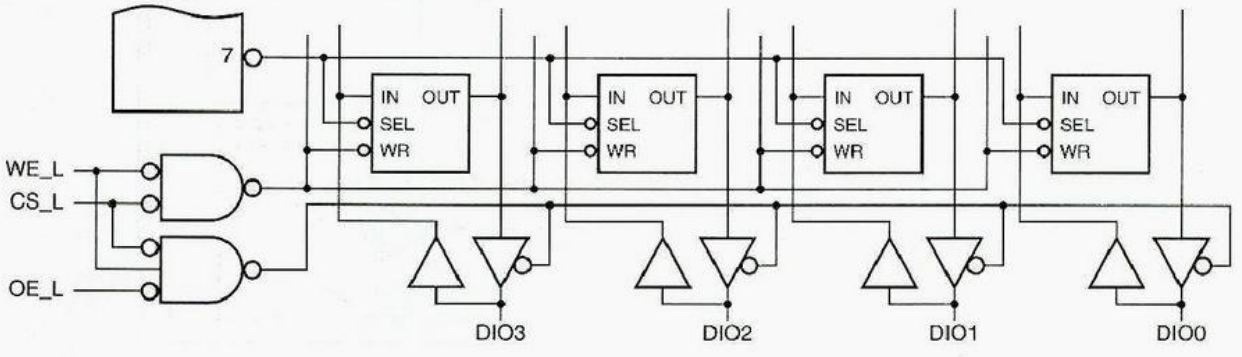
Las SRAM de la figura 10-24 tienen buses de datos bidireccionales: esto es, utilizan las mismas terminales de datos tanto para lectura como para escritura. Esto necesita de un ligero cambio en su lógica de control interna, mostrada en la figura 10-25. El separador de salida se deshabilita en forma automática siempre que WE_L es asertiva, incluso si OE_L es asertiva. Sin embargo, los parámetros de tiempo y requerimientos para operaciones de lectura y escritura son prácticamente idénticos a los que describimos en la subsección anterior.

Un uso común de las SRAM es el almacenamiento de datos en pequeños sistemas de microprocesador, a menudo en aplicaciones "integradas" (teléfonos, tostadores, parachoques eléctricos, etc.). Las computadoras de propósito general utilizan con más frecuencia las DRAM, que se presentan en la figura 10-33, debido a que su densidad es mayor y su costo por bit es más bajo.

Las SRAM muy rápidas se utilizan con frecuencia en las memorias "caché" de las computadoras de alto desempeño para almacenar las instrucciones y datos empleados con frecuencia. De hecho, la "necesidad de velocidad" en las aplicaciones de memoria caché de la PC conducen al desarrollo y utilización de una interfaz SRAM temporizada y más rápida. Las SRAM estándar en esta subsección se denominan ahora *SRAM asincrónicas* para distinguirlas del nuevo estilo que se discute a continuación.

SRAM asincrónicas

Figura 10-25 Control de separador de salida en una SRAM con un bus de datos bidireccional.



*10.3.5 SRAM sincrónica

SRAM sincrónica (SSRAM)

Una nueva variedad de SRAM, denominada *SRAM sincrónica (SSRAM)*, todavía utiliza latchs en forma interna pero tiene una interfaz temporizada para control, direcciones y datos. Como se muestra en la figura 10-26, los registros internos disparados por flanco AREG y CREG son colocados en las trayectorias para dirección y control. Como resultado, una operación que se establece antes del flanco ascendente del reloj se realiza en forma interna durante un periodo del reloj posterior. El registro INREG captura los datos de entrada para las operaciones de escritura y, dependiendo si el dispositivo tiene salidas "canalizadas" o "de flujo continuo", se proporciona (o no) el registro OUTREG para retener los datos de salida de una operación de lectura.

SSRAM de escritura diferida con salidas de flujo continuo

La primera variedad de SSRAM que fue presentada fue la *SSRAM de escritura diferida con salidas de flujo continuo*. Para una operación de lectura, que se muestra en la figura 10-27(a), las entradas de control y dirección se muestrean en el flanco ascendente del reloj, y el registro de dirección interno AREG es cargado sólo si ADS_L es afirmada. Durante el siguiente periodo de reloj, se tiene acceso al arreglo de SRAM interno y los datos de lectura se entregan a las terminales del bus de datos DIO del dispositivo. El dispositivo

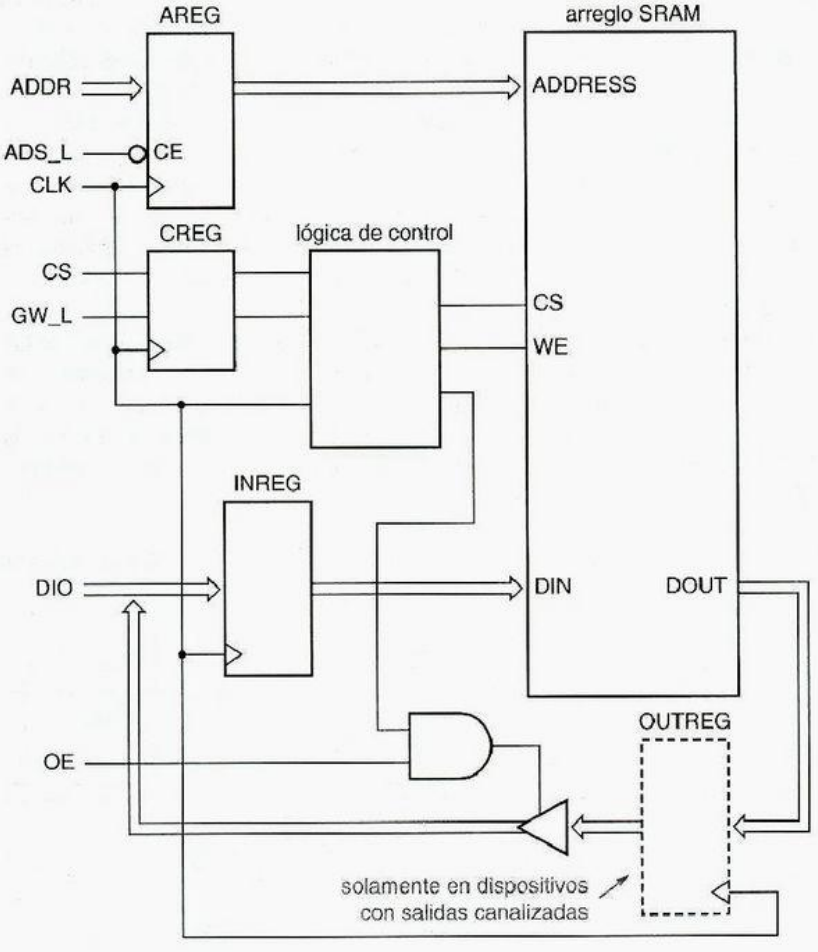


Figura 10-26
Estructura interna de una SRAM sincrónica.

también soporta un modo de ráfaga, en que los datos en secuencia de direcciones se leen. En este modo AREG se comporta como un contador, eliminando la necesidad de aplicar una nueva dirección en cada ciclo. (Las señales de control que soportan el modo de ráfaga no se muestran en las figuras 10-26 o 10-27.)

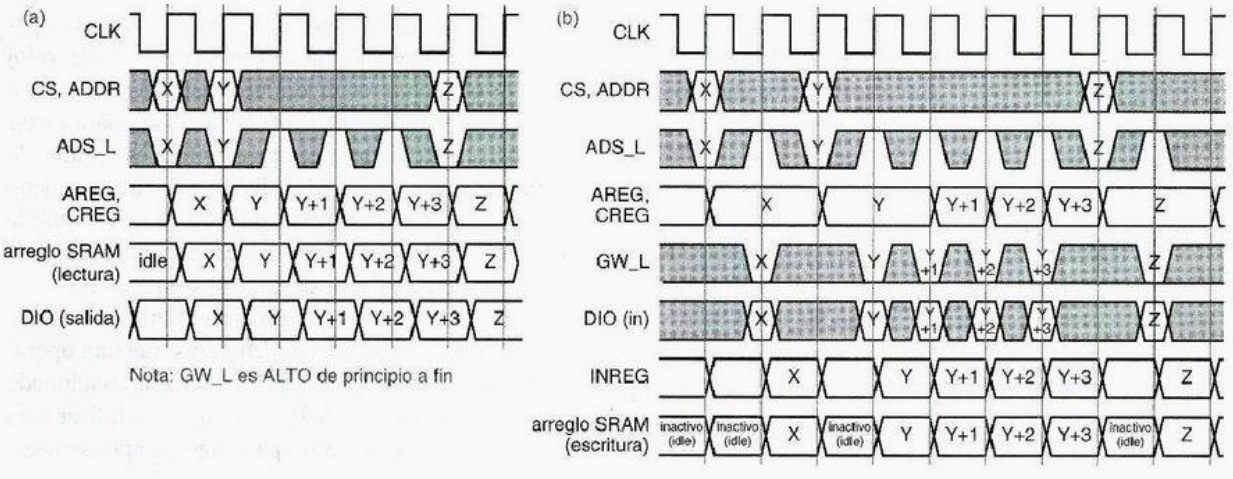
Para una operación de escritura, mostrada en b), los datos de escritura se almacenan temporalmente en un registro INREG en el chip, el cual se muestrea un tic de reloj *después* de que se carga el registro de dirección. Por lo tanto, ADS_L debe inhibirse por al menos un tic después de cargar la dirección, de modo que la dirección en AREG sea todavía válida cuando la escritura tenga lugar. La escritura se presenta durante el periodo de reloj que sigue al flanco en el cual la señal de control de "escritura global" GW_L es afirmada. Como en el caso de la lectura, el dispositivo tiene un modo de ráfaga donde una secuencia de direcciones puede escribirse sin suministrar una nueva dirección.

Observe que el protocolo de "escritura diferida" imposibilita la escritura en dos direcciones diferentes y no secuenciales para períodos sucesivos de reloj; el arreglo SRAM está inactivo en un período de reloj entre las operaciones de escritura (excepto en el modo ráfaga). Desde el punto de vista de las capacidades internas del chip, no se necesita este comportamiento. Sin embargo, el protocolo de escritura diferida se diseñó de esta forma para igualar los protocolos de bus de los microprocesadores que utilizan las SSRAM en sus subsistemas de memoria caché.

SSRAM de escritura diferida con salidas canalizadas

Una *SSRAM de escritura diferida con salidas canalizadas* es semejante a la versión anterior, excepto por que un registro OUTREG se coloca entre la salida del arreglo SRAM y la salida del dispositivo para operaciones de lectura. Como se muestra en la figura 10-28, esto retarda los datos de la salida de lectura en las terminales del dispositivo hasta el inicio del siguiente periodo de reloj, pero también proporciona el siguiente beneficio: ahora los datos son válidos para casi todo el ciclo del reloj. El ciclo de escritura tiene el mismo comportamiento que con las salidas de flujo continuo. Comparadas con las salidas de flujo continuo, las salidas canalizadas proporcionan un tiempo de establecimiento

Figura 10-27 Comportamiento de temporización para una SRAM de SCLK diferida con salidas de flujo continuo: (a) operaciones de lectura; (b) operaciones de escritura.



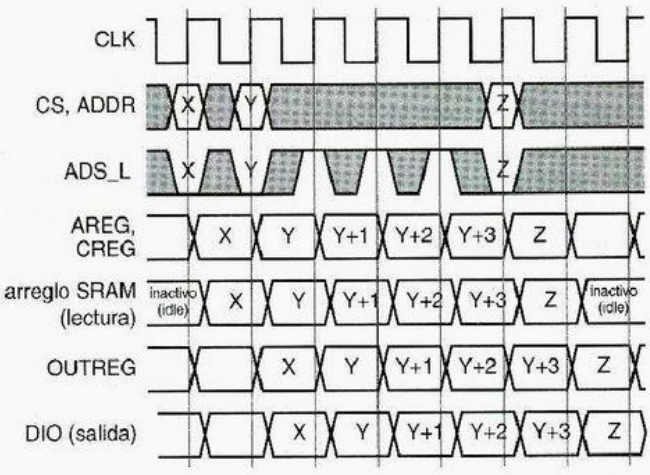


Figura 10-28
Comportamiento de temporización de lectura para una SSRAM de escritura diferida con las salidas canalizadas.

mucho mejor para el dispositivo que recibe los datos de lectura, y, por consiguiente, puede permitir operaciones a frecuencias más altas del reloj.

Como mostramos en la figura 10-26, las SSRAM convencionales comparten las mismas terminales tanto para los datos de entrada como para los datos de salida. Durante un periodo específico del reloj, las terminales E/S de datos se pueden utilizar para leer o escribir, pero no para ambas cosas. Si se estudia el patrón de uso del bus de datos y del arreglo de SRAM en ambos estilos de SSRAM de escritura diferida, encontrará casos donde no es posible iniciar una lectura un ciclo de reloj después de iniciar una escritura o viceversa, debido al conflicto de recursos (véase el ejercicio 10.32). De este modo, las SSRAM de escritura diferida sufren una *penalización de carga y descarga*, un ciclo del reloj en el cual el arreglo interno SRAM debe estar inactivo cuando una lectura esté seguida de una escritura o viceversa.

penalización de carga y descarga

La penalización de carga y descarga es eliminada en las denominadas *SSRAM ZBT (zero-bus-turn-around, de carga y descarga de bus cero)*. La temporización para una *SSRAM ZBT con salidas de flujo continuo* se muestra en la figura 10-29. El tipo de operación (lectura o escritura) se selecciona mediante una señal de control R/$\overline{W}$ que muestrea en el mismo flanco del reloj que la dirección. Sin tener en cuenta si la operación es de lectura o escritura, el bus DIO es utilizado durante el siguiente ciclo del reloj para transferir los datos de lectura o escritura. Como resultado, no existe un conflicto en el uso del bus de datos, mientras que OE se controla en la forma adecuada para evitar conflicto en el bus entre los ciclos sucesivos. Sin embargo, si una escritura es seguida de una lectura, ambas operaciones querrían emplear el arreglo SRAM durante el mismo ciclo del reloj. Para evitar este conflicto de recursos, la operación de escritura es diferida hasta el siguiente ciclo SRAM disponible. Esta oportunidad se presenta cuando alguna operación de escritura o ninguna otra se inicia en las líneas de control y dirección.

SSRAM de carga y descarga de bus cero (ZBT)

Aunque una SSRAM ZBT puede usar el acceso al arreglo SRAM interno en cada ciclo del reloj, esta mejora del desempeño conlleva un precio. Mientras que una operación de escritura está pendiente, la dirección de escritura y la información relacionada deben almacenarse en otro registro, WAREG, puesto que AREG se vuelve a utilizar para otras operaciones; esto sacrifica área del chip. Más significativa para algunas aplicaciones,

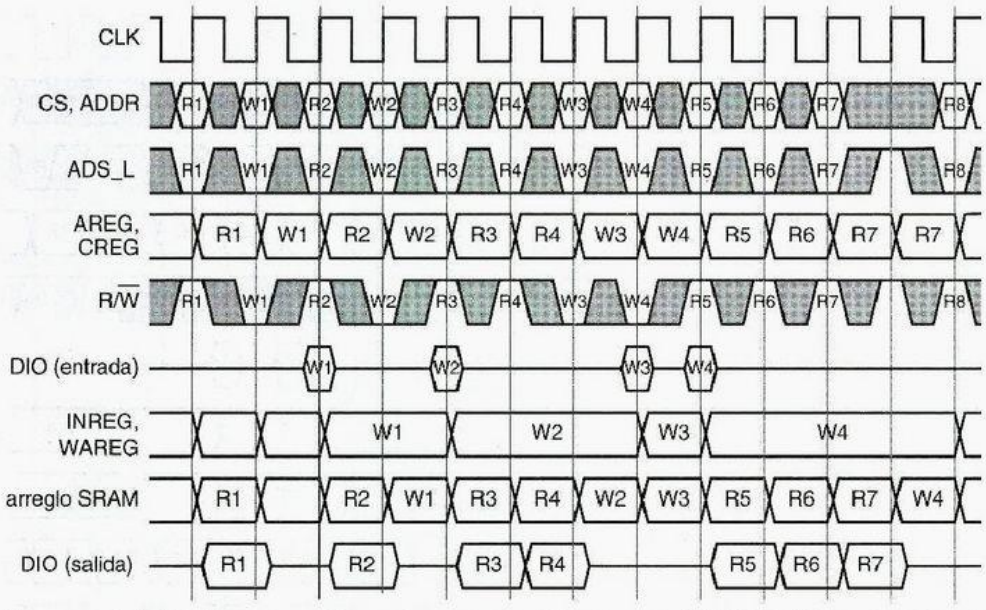


Figura 10-29 Comportamiento de temporización para una SSRAM ZBT con salidas de flujo continuo.

una operación de escritura puede ser diferida de manera indefinida si está seguida de inmediato por una serie continua de operaciones de lectura. Esta anomalía puede requerir de un delicado diseño de controlador para detectar el caso donde una de estas operaciones de lectura intenta tener acceso a la dirección que acaba de escribirse, puesto que el valor almacenado en el arreglo SRAM está ¡"pasado"!

Una *SSRAM ZBT con salidas canalizadas* agrega OUTREG a la trayectoria de datos de lectura pero de otra manera es similar al dispositivo anterior. En este dispositivo, tanto las lecturas como las escrituras utilizan el bus DIO durante el segundo ciclo del reloj siguiendo el flanco del reloj en el que la operación fue iniciada. Como en el dispositivo previo, las escrituras al arreglo interno SRAM son diferidas hasta un ciclo disponible, de modo que las lecturas pueden tener precedencia. La temporización se muestra en la figura 10-30. Como se implica por la temporización, se necesitan dos niveles de registros internos para escribir direcciones y datos, ya que se pueden diferir hasta dos escrituras mientras se presenta una secuencia de lecturas.

Los protocolos de acceso a una SRAM sincrónica son muy convenientes en los sistemas de alta velocidad. Por ejemplo, las entradas de dirección, control y escritura pueden aplicarse con tiempos de establecimiento y retención más o menos convencionales con respecto al reloj del sistema, y los datos de lectura en terminales de salida canalizada se encuentran disponibles para casi un ciclo completo de reloj. Lo que es más importante, el diseñador no tiene que preocuparse acerca de las trayectorias de temporización y de los circuitos difíciles que de otra manera serían necesarios para habilitar el funcionamiento convencional tipo latch de la SRAM.

Las SSRAM estaban disponibles en 1999 con frecuencias de reloj tan altas como 166 MHz. Advierta que entre los cuatro estilos de SSRAM que hemos descrito, ninguno es "el mejor". La mejor SSRAM es aquella que mejor se ajusta al protocolo de bus y otros requerimientos del sistema en el que se está utilizando.

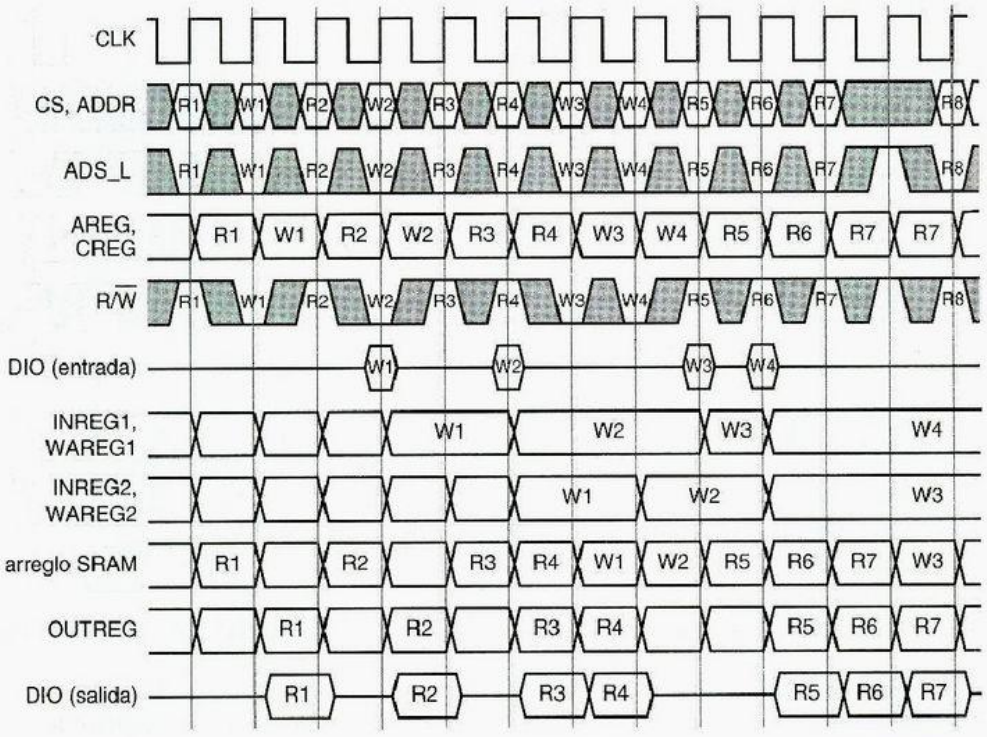


Figura 10-30 Comportamiento de temporización para una SSRAM ZBT con salidas canalizadas.

10.4 RAM dinámica

La celda de memoria básica en una SRAM, un *latch D* requiere de cuatro compuertas en un diseño discreto, y de cuatro a seis transistores en un chip LSI de SRAM de diseño personalizado. A fin de construir RAM con mayor densidad (más bits por chip), los diseñadores de chips inventaron celdas de memoria que emplean una cantidad muy pequeña: un transistor por bit.

10.4.1 Estructura de una memoria dinámica tipo RAM

RAM dinámica (DRAM)

No es posible construir un elemento flip-flop con sólo un transistor. En su lugar, las celdas de memoria en una *RAM dinámica (DRAM)* almacenan información en un capacitor diminuto al que se tiene acceso a través de un transistor MOS. La figura 10-31 muestra la celda de almacenamiento para un bit de una memoria DRAM, a la cual se tiene acceso estableciendo la línea de palabra a un voltaje ALTO. Para almacenar un 1, se asigna un voltaje ALTO en la línea del bit, lo que carga el capacitor a través del transistor de "encendido". Para almacenar un 0, un voltaje BAJO en la línea del bit descarga el capacitor.

precarga (carga previa)

amplificador de sensibilidad

Para leer una celda de DRAM, la línea de bit es primero *precargada* con un voltaje de una magnitud entre los valores ALTO y BAJO, y entonces la línea de palabra se establece en ALTO. Dependiendo si el voltaje del capacitor es ALTO o BAJO, la línea de bit precargada se mueve ligeramente más arriba o ligeramente más abajo. Un *amplificador de sensibilidad* detecta este pequeño cambio y recupera un 1 o un 0 de acuerdo con ello. Advierta que la

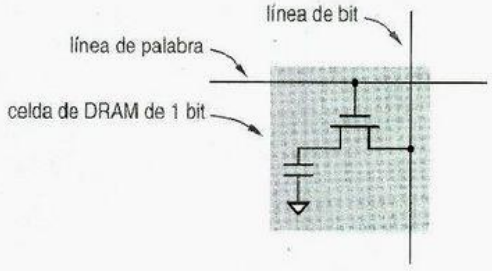


Figura 10-31
Celda de almacenamiento para un bit en una memoria DRAM.

lectura de una celda destruye el voltaje original que se almacena en el capacitor, de modo que los datos recuperados deben escribirse de regreso a la celda después de la lectura.

El capacitor en una celda DRAM tiene una capacitancia muy pequeña, pero el transistor MOS que tiene acceso a él tiene una impedancia muy alta. Por lo tanto, para un voltaje ALTO se necesita un tiempo más o menos largo (muchos milisegundos) para descargar al punto que parece más como un voltaje BAJO. Mientras tanto, el capacitor almacena un bit de información.

Por supuesto, utilizar una computadora no sería divertido si tuviera que reinicializarla cada "x" milisegundos porque desaparece el contenido de su memoria (el típico comportamiento de Windows de todos modos). En consecuencia, los sistemas de memoria basados en DRAM emplean *lazos de actualización* para "refrescar" todas las celdas de memoria en forma periódica, una vez cada cuatro milisegundos en las primeras DRAM. Esto involucra la lectura secuencial del contenido algo degradado de cada celda en un latch D, y la actualización de un valor bastante sólido BAJO o ALTO a partir del latch. La figura 10-32 ilustra el estado eléctrico de una celda después de una escritura y una secuencia de operaciones de actualización.

lazo de actualización

Las primeras DRAM, que se comercializaron a principios de los años setenta, contenían sólo 1,024 bits, pero las DRAM modernas están disponibles con 256 *megabits* o más. Si se tiene que refrescar cada celda, una a la vez, en cuatro milisegundos, tendríamos un problema (lo que resultaría en mucho menos de 1 ns por celda, y no incluye tiempo para operaciones de lectura y escritura útiles). Por fortuna, como demostraremos, las memorias DRAM se organizan utilizando arreglos de dos dimensiones, y una sola operación refresca un renglón completo del arreglo. Las primeras memorias DRAM tenían 256 renglones, que requerían 256 operaciones de actualización cada cuatro milisegundos, o aproximadamente una cada 15.6 μs. Los arreglos más recientes tienen 4096 renglones pero necesitan ser refrescados sólo una vez cada 64 ms, lo que todavía da como resultado en un renglón cada 15.6 μs. Una operación de actualización por lo regular consume menos de 100 ns, de modo que la memoria DRAM está disponible para operaciones de lectura y escritura útiles alrededor del 99% del tiempo.

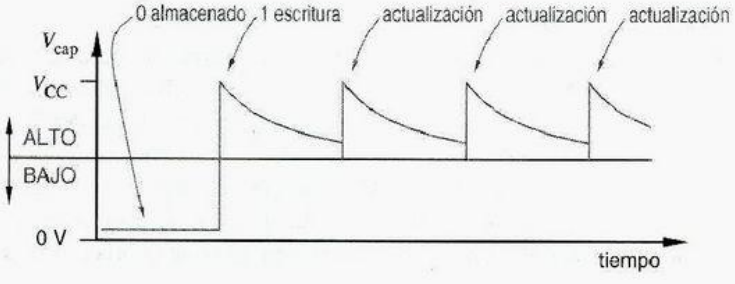


Figura 10-32
Voltaje almacenado en una celda DRAM después de operaciones de escritura y actualización.

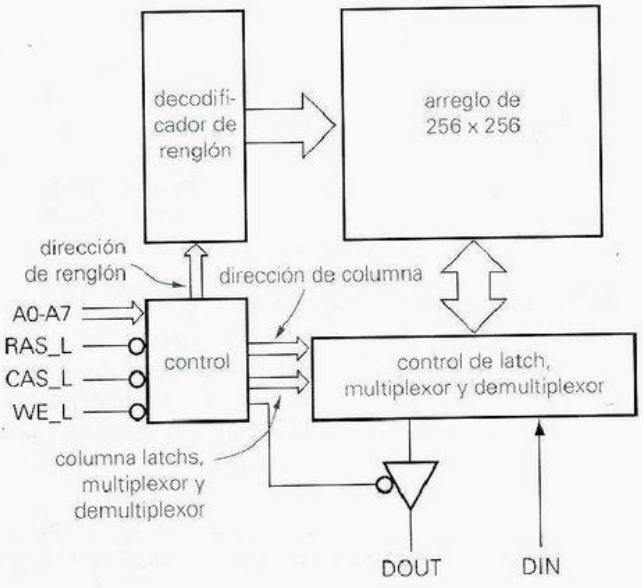


Figura 10-33
Estructura interna de una DRAM de 64K × 1.

entradas de dirección multiplexadas

RAS_L ("row address strobe", sincronía de dirección de renglón)

CAS_L ("column address strobe", sincronía de dirección de columna)

La figura 10-33 es un diagrama de bloques de la estructura interna de una DRAM de 64K × 1. El arreglo lógico tiene 64K × 1 bits, pero el arreglo físico es cuadrado, con 256 × 256 bits. Aunque la memoria tiene 64K localidades, el chip tiene sólo ocho *entradas de dirección multiplexadas*. Una dirección completa de 16 bits se presenta al chip en dos pasos controlados por dos señales: *RAS_L ("row address strobe", sincronía de dirección de renglón)* y *CAS_L ("column address strobe", sincronía de dirección de columna)*. El multiplexado de las entradas de dirección ahorra terminales, algo importante para el diseño compacto de sistemas de memoria, y también se ajusta de modo bastante natural con los métodos de acceso a memorias DRAM de dos pasos que describiremos con brevedad.

Las memorias DRAM que son más grandes tienen arreglos mayores que con frecuencia son múltiples. Una ventaja de los arreglos múltiples es facilitar los problemas de diseño eléctricos y físicos que se presentarían con un arreglo extremadamente grande. Pero aún más importante es el paralelismo que puede ocurrir con los arreglos múltiples. Como veremos en la subsección siguiente, el funcionamiento de la memoria DRAM es mucho más complicado que el de una memoria SRAM. Aprovechando los arreglos múltiples en las DRAM más grandes de alta velocidad, un controlador moderno de memoria DRAM puede realizar varias operaciones en paralelo, por ejemplo, completar una operación de escritura en un arreglo mientras que se inicia una operación de lectura en otro. Esto incrementa el rendimiento efectivo de la memoria.

10.4.2 Temporización de la RAM dinámica

Existen muchos escenarios diferentes de temporización para diferentes tipos y operaciones de DRAM. En esta sección describiremos los lazos más comunes para la DRAM convencional y los relacionaremos con la estructura interna del dispositivo. El aspecto más sorprendente de la temporización de DRAM convencional es que no hay reloj. En su lugar, las operaciones de DRAM se inician y completan en ambos flancos de ascenso y descenso o caída de RAS_L y CAS_L. ¡Esto es en realidad una temporización muy delicada, pero la industria de algún modo ha vivido con ella por más de 25 años!

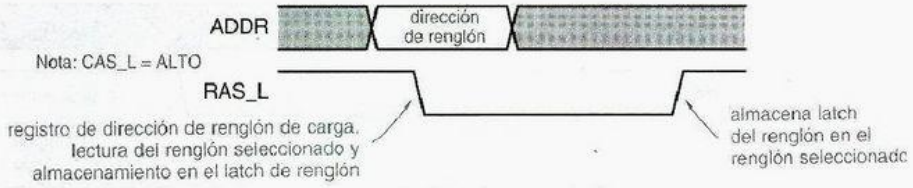


Figura 10-34
Temporización de ciclo de actualización de sólo RAS de DRAM.

La temporización para un *lazo de actualización de sólo RAS* se muestra en la figura 10-34. Este lazo se utiliza para refrescar un renglón de memoria sin leer o escribir en realidad ningún dato en las terminales externas del chip DRAM. El lazo comienza cuando una dirección de renglón se aplica a las entradas de dirección multiplexadas (ocho bits en el caso de una DRAM de 64K×1) y RAS_L es afirmada. La DRAM almacena la dirección de renglón en un *registro de dirección de renglón* en el flanco de caída de RAS_L y lee el renglón seleccionado del arreglo de memoria en un *latch de renglón* en el chip. Cuando RAS_L es negada, el contenido del renglón se escribe de vuelta desde el latch de renglón. Para refrescar una DRAM de 64K × 1 por entero, el diseñador del sistema debe asegurar que 256 lazos de esta naturaleza, utilizando en su totalidad las 256 direcciones de renglón posibles, se ejecutan cada cuatro milisegundos. Un contador externo de 8 bits puede utilizarse para generar las direcciones de renglón, y se emplea un temporizador para iniciar un lazo de actualización una vez cada 15.6 μs.

lazo de actualización sólo RAS

registro de dirección de renglón

latch de renglón

Un *lazo de lectura*, mostrado en la figura 10-35, comienza como un lazo de actualización, donde un renglón seleccionado se lee dentro del latch de renglón. A continuación, se aplica una dirección de columna a las entradas de dirección multiplexadas y se almacena en un *registro de dirección de columna* en el chip en el flanco de caída de CAS_L. La

ciclo de lectura

registro de dirección de columna

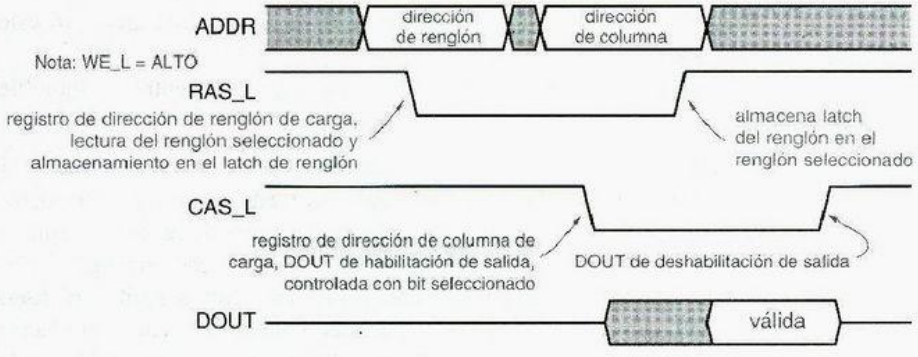


Figura 10-35
Temporización de ciclo de lectura DRAM.

TEMPORIZACIÓN DELICADA

Usted puede haber notado que los intervalos válidos de direcciones de renglón y columna en la figura 10-35 están sesgados hacia la derecha con respecto a los flancos de caída de RAS_L y CAS_L, lo que carga las direcciones en registros disparados por flanco en el chip. En general, las memorias DRAM requieren un tiempo de establecimiento corto para las entradas de dirección (con frecuencia de 0 ns), pero un requerimiento de tiempo de retención positivo relativamente largo (7-20 ns). Esto puede ocasionar algunos problemas en el diseño de un sistema sincrónico distinto que utiliza un reloj simple y flip-flops con cero tiempos de retención, y con frecuencia es resuelto mediante desagradables métodos que involucran líneas de retardo o el uso del flanco opuesto del reloj. Ésta es una de las razones por las cuales muchos diseñadores consideran el diseño de sistema DRAM como una delicada magia negra.

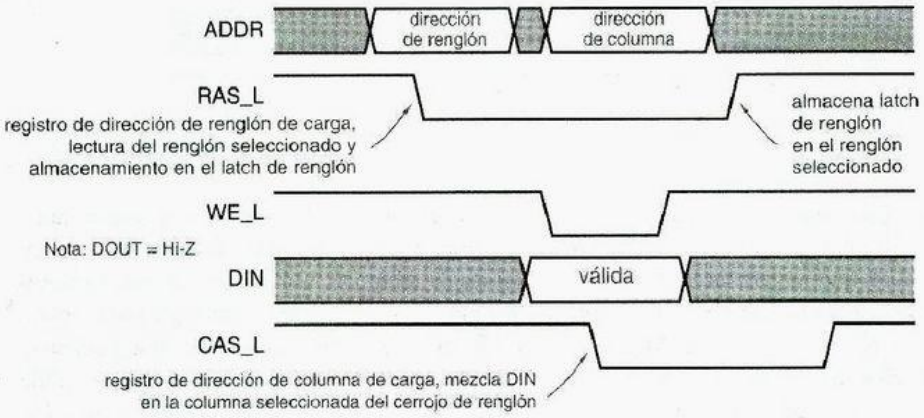


Figura 10-36
Temporización de ciclo de escritura de DRAM.

dirección de columna se emplea para seleccionar un bit del renglón que se acaba de leer, el cual se tiene disponible en la terminal DOUT de la DRAM. Una terminal de tres estados, DOUT es habilitada para salida mientras CAS_L es asertiva. Mientras tanto, todo el renglón se escribe de regreso en el arreglo tan pronto como RAS_L es negada.

lazo de escritura WE_L (write enable, habilitación de escritura)

Un *lazo de escritura*, mostrado en la figura 10-36, también se inicia como un lazo de actualización o lectura. Sin embargo, *WE_L (write enable, habilitación de escritura)* debe ser asertiva antes de que CAS_L lo sea para seleccionar un lazo de escritura. Esto se logra deshabilitando DOUT para el resto del lazo, aun cuando CAS_L será afirmada en forma posterior. Una vez que el renglón seleccionado se lee en el latch de renglón, WE_L también obliga al bit de entrada en DIN a mezclarse en el latch de renglón, en la posición de bits seleccionada por la dirección de columna. Cuando el renglón se vuelve a escribir en el arreglo durante el flanco ascendente de RAS_L, contiene un nuevo valor en la columna seleccionada.

Varios otros tipos de lazos, no mostrados en la figura, se encuentran disponibles en las DRAM típicas:

lazo de actualización CAS-antes-de-RAS

- Un *ciclo de actualización CAS antes de RAS* realiza una operación de actualización sin requerir que se suministre una dirección de renglón por medio de un contador externo. En su lugar, el chip DRAM utiliza un contador de dirección de renglón interno. Si CAS_L es afirmada antes de que RAS_L lo sea, la DRAM refresca el renglón seleccionado por el contador y entonces incrementa el contador. Esto simplifica el diseño del sistema de DRAM, eliminando el contador externo de actualización, y reduciendo de tres (renglón, columna, actualización) a dos el número de fuentes multiplexadas que deben controlar las entradas de dirección de la DRAM.

lazo de lectura-modificación-escritura

- Un *lazo de lectura-modificación-escritura* comienza como un ciclo de lectura y permite que los datos sean leídos en DOUT cuando CAS_L es asertiva. Sin embargo, WE_L puede ser afirmada posteriormente para venir a escribir nuevos datos en la misma localidad.

lazo de lectura en modo de página

- Un *lazo de lectura en modo de página* permite que hasta un renglón entero ("página") de datos sea leído sin repetir todo el lazo RAS-CAS. Puesto que el renglón entero ya se ha almacenado en el cerrojo del renglón, este lazo simplemente requiere que CAS_L sea excitado a nivel BAJO múltiples veces mientras RAS_L permanece afirmada de manera continua. Una nueva dirección de columna es proporcionada y un nuevo bit se hace disponible en DOUT en cada flanco de caída de CAS_L.

Este lazo proporciona un acceso a memoria mucho más rápido durante una secuencia de lecturas a direcciones consecutivas o de otro modo, cercanas; tales secuencias se presentan con frecuencia en microprocesadores cuando se obtienen instrucciones o se llenan memorias caché.

- De manera similar, un *lazo de escritura en modo de página* permite que sean escritos múltiples bits de un renglón con un solo ciclo RAS_L y múltiples ciclos CAS_L.

lazo de escritura en modo de página

Otros dos modos de acceso múltiple, *modo de columna estática y modo de medio byte (nibble)*, fueron suministrados alguna vez en ciertos chips DRAM, pero llegaron a ser obsoletos con el dominio comercial de las denominadas *DRAM de salida de datos extendida (EDO, extended-data-out)*. En estos dispositivos, CAS_L ya no controla la habilitación de salida durante los lazos de lectura; en su lugar; se proporciona una entrada de control OE_L por separado. Esto es importante para lazos de lectura en modo página de alta velocidad, en la medida que proporciona datos de salida válidos para un extenso periodo. En vez de apagarse entre pulsos sucesivos CAS_L, la salida se deja encendida en forma continua (aunque controlada por OE_L) y los datos de lectura en modo de página son válidos desde el principio de un pulso CAS_L hasta el principio del siguiente.

modo de columna estática

modo de medio byte (nibble)

DRAM de salida de datos extendida (EDO)

10.4.3 DRAM sincrónicas

El protocolo de acceso basado en flanco RAS/CAS de las memorias DRAM convencionales no sólo es delicado, sino que es difícil hacer que vaya rápido y todavía satisfaga los márgenes de temporización cuando se conecta al resto de un sistema. De esta manera, al principio de la década de 1990 vio el desarrollo de las memorias *DRAM sincrónicas (SDRAM)* que emplean una interfaz de temporización más convencional, y hacia finales de la misma década estos dispositivos dominaban el mercado de las memorias para PC.

DRAM sincrónica (SDRAM)

Las DRAM sincrónicas retienen el estilo de direccionamiento multiplexado de las DRAM convencionales (las direcciones de renglón y columna se entregan en dos pasos). Sin embargo, las señales de control de una memoria SDRAM, además de sus entradas de dirección, se muestrean sólo en el flanco ascendente de una señal del reloj común (CLK) con una frecuencia tan alta como 133 MHz. Además, una SDRAM tiene una señal de habilitación del reloj (CKE) de modo que las entradas son ignoradas si CKE no es asertiva. Para operaciones de escritura, los datos se muestrean en el flanco del reloj, y para lecturas, los datos de salida son entregados en el flanco del reloj.

Del mismo modo que una memoria DRAM convencional, una SDRAM necesita varios pasos internos para efectuar una operación, y esto significa múltiples tics del reloj de manera externa. Internamente, las memorias SDRAM tienen múltiples bancos DRAM, por lo común cuatro, y pueden tener múltiples operaciones pendientes de manera simultánea.

Para cada tic del reloj, las señales de control de SDRAM RAS_L, CAS_L y WE_L son interpretadas como una "palabra de mando", en contraposición a tener significado individual. Al mismo tiempo, los bits de dirección de orden superior son interpretados como un "selector de banco" para indicar a cuál banco aplicar un mando. Por ejemplo, un controlador inteligente de SDRAM puede emplear cuatro tics del reloj para iniciar las operaciones de lectura en cuatro diferentes bancos, y posteriormente regresar y leer los resultados uno por cada tic del reloj a medida que se completan.

La temporización interna en una memoria SDRAM se obtiene de la entrada de reloj CLK. Por lo regular, una señal RAS para el arreglo interno será activada inmediatamente después del tic del reloj cuando sea dada una orden de lectura o escritura. Para

satisfacer los requerimientos de temporización interna, el chip genera una señal interna CAS un tic del reloj después. Determinar cuántos tics después es una función de la frecuencia CLK y del grado de velocidad del mismo chip de memoria SDRAM. Para facilitar diferentes requerimientos, el retardo RAS-a-CAS, conocido como la latencia CAS, es programable. Éste y otros parámetros importantes de operación deben descargarse en una SDRAM al momento de su inicialización. La descarga es muy fácil; el comando "cargar parámetros" es reconocido cuando las señales de control RAS_L, CAS_L y WE_L son todas asertivas en forma simultánea, y los parámetros mismos son transferidos en las líneas de dirección.

10.5 Dispositivos lógicos programables complejos

Desde su introducción hace ya varios años, los dispositivos lógicos programables como el 16V8 y el 22V10 han sido caballitos de batalla muy flexibles en el diseño digital. A medida que la tecnología de los CI avanza, ha habido gran interés en la creación de arquitecturas PLD más grandes para aprovechar la densidad creciente de los chips. La pregunta es, ¿por qué los fabricantes no se limitan a escalar las arquitecturas existentes?

Por ejemplo, si las densidades de la memoria DRAM se incrementaron por un factor de 64 en los últimos 10 años, ¿por qué no podrían los fabricantes escalar el 16V8 para crear un "128V64"? Un dispositivo de esa naturaleza tendría 64 terminales de entrada, 64 terminales de E/S y algún número de términos de producto de 128 variables (digamos, 8) para cada una de sus 128 macroceldas lógicas. Podría combinar las funciones de una colección más grande de 16V8 y ofrecer un impresionante rendimiento y flexibilidad al utilizar cualquier entrada en cualquier función de salida.

¿O no? Sí, un 128V64 sería muy flexible, pero no tendría muy buen desempeño. A diferencia del 16V8, que tiene 32 entradas (16 señales y sus complementos) por término AND, este dispositivo tendría 256. Debido a los efectos capacitivos, las corrientes de fuga y cosas por el estilo, una estructura alambrada AND así de grande sería por lo menos ocho veces más lenta que el arreglo AND del 16V8.

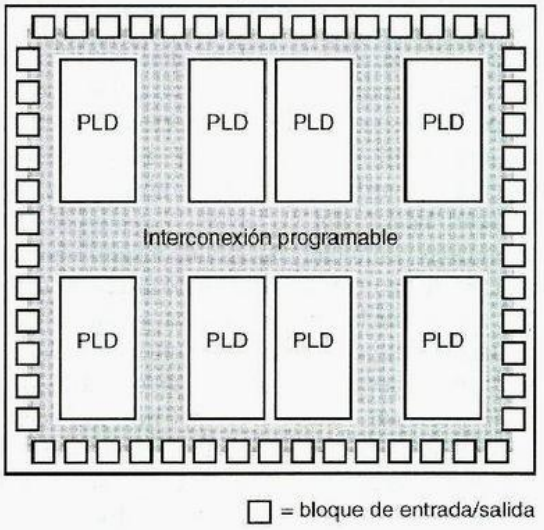


Figura 10-37
Arquitectura CPLD general.

Lo peor, desde el punto de vista de un fabricante, es que un 128V64 no haría un uso efectivo en cuanto al costo del área del chip. Emplearía aproximadamente 64 veces el área de chip de un 16V8, pero proporcionaría el mismo número de entradas y salidas que solamente ocho 16V8. Es decir, para n veces tanta lógica (en términos de entradas, salidas y términos AND), el 128V64 utilizaría n^2 área de chip equivalente. Desde el punto de vista de eficiencia de uso del silicio, un diseñador astuto haría mejor si dividiera una función deseada en ocho 16V8, si tal partición fuera posible.

De aquí es de donde viene la idea de los dispositivos lógicos programables complejos (CPLD). Como se muestra en la figura 10-37, un CPLD es sólo una colección de PLD individuales en un simple chip, en compañía de una estructura de interconexión programable que permite que los PLD sean conectados entre sí en el chip de la misma manera que un diseñador astuto puede hacerlo con PLD discretos al exterior del chip. Aquí, el área del chip para n veces una lógica equivalente es sólo n veces el área de un PLD simple más el área de la estructura de interconexión programable.

Diferentes fabricantes han tomado muchos enfoques diferentes para la arquitectura general mostrada en la figura. Las áreas en las cuales difieren incluyen los PLD individuales (arreglos AND y macroceldas), los bloques de entrada/salida y la interconexión programable. Estudiaremos cada una de estas áreas en el resto de esta sección, aplicando la arquitectura CPLD de la serie 9500 de Xilinx como un ejemplo representativo.

10.5.1 La familia CPLD XC9500 de Xilinx

La serie XC9500 de Xilinx es una familia de CPLD con una arquitectura similar pero números variantes de terminales de entrada/salida (E/S) externas y PLD internos (que Xilinx llama bloques de función, FB, función block). Como veremos más adelante cada PLD interno tiene 36 entradas y 18 macroceldas y salidas y puede ser llamado un "36V18". Como se muestra en la tabla 10-8, los dispositivos en la familia son nombrados de acuerdo con el número de macroceldas que contengan. El más pequeño tiene 2 FB y 36 macroceldas, mientras que el más grande tiene 16 FB y 288 macroceldas.

Otra característica importante de ésta y la mayoría de las familias CPLD es que un chip dado, tal como el XC95108, se encuentra disponible en varios encapsulados diferentes. Esto es importante no sólo para acomodar diferentes prácticas de fabricación, sino también para proporcionar alguna selección y ahorros potenciales en el número de terminales de E/S externas proporcionadas. En la mayoría de las aplicaciones, no es necesario que todas las señales internas de una máquina de estado o subsistema sean visibles y se utilicen por el resto del sistema.

Así, aun cuando el XC95108 tiene 108 macroceldas internas, las salidas de por lo menos 69 de ellas se pueden conectar en forma externa en la versión PLCC de 84 terminales del dispositivo. De hecho, muchas de las 69 terminales de E/S se utilizarían como entradas, en cuyo caso se verán menos salidas en forma externa. Esto está bien; las salidas de macrocelda restantes todavía se pueden utilizar internamente, puesto que pueden conectarse de modo interno a través de la interconexión programable del CPLD. En ocasiones las macroceldas cuyas salidas solamente son utilizables en forma interna se denominan *macroceldas enterradas*.

macrocelda enterrada

Tabla 10-8 Bloques de función y terminales de E/S externas en los CPLD serie 9500 de Xilinx.

	Número de parte					
	XC9536	***XC9572***	***XC95108***	***XC95144***	***XC95216***	***XC95288***
FBs / macroceldas	2 / 36	4 / 72	6 / 108	8 / 144	12 / 216	16 / 288
Encapsulado	***Terminales de E/S del dispositivo***					
VQFP de 44 terminales	34					
PLCC de 44 terminales	34	34				
CSP de 48 terminales	34					
PLCC de 84 terminales		69	69			
TQFP de 100 terminales		72	81	81		
PQFP de 100 terminales		72	81	81		
PQFP de 160 terminales			108	133	133	
HQFP de 208 terminales					166	168
BGA de 352 terminales					166	192

Otra importante dimensión de la tabla 10-8 es la horizontal. Todos, excepto dos de los encapsulados, respaldan por lo menos dos diferentes dispositivos en el mismo encapsulado y, a medida que se producen, con terminales de salida compatibles. Esto es un salvavidas en la creación de cambios de último minuto. Por ejemplo, supongamos que usted fuera a destinar un diseño a un XC9572 en un PLCC de 84 terminales. Puede encontrar que las 69 terminales de E/S del dispositivo son bastante adecuadas. A usted le gustaría utilizar el XC9572 por su bajo costo, pero tal vez se sienta un poco nervioso si su diseño inicial emplea 68 de las 72 macroceldas internas disponibles (¡sé que yo lo estaría!). Basado en la tabla 10-8, puede dormir a gusto sabiendo que si los arreglos a los errores o cambios de especificación del diseño necesitan un diseño más complejo internamente, usted siempre puede ascender al XC95108 en el mismo encapsulado y conseguir otras 36 macroceldas.

La figura 10-38 es un diagrama de bloques de la arquitectura interna de un CPLD típico de la familia XC9500. Cada terminal de E/S externa puede ser empleada como una entrada, una salida o una terminal bidireccional de acuerdo a la programación del dispositivo, como se discute posteriormente. Las terminales en la parte inferior de la figura también pueden utilizarse para propósitos especiales. Cualquiera de las tres terminales puede emplearse como "relojes globales" (GCK, global clocks); como veremos más tarde, cada macrocelda puede ser programada para utilizar una entrada del reloj seleccionada. Una terminal puede emplearse como un "establecimiento/restablecimiento global" (GSR, global set/reset); de nueva cuenta, cada macrocelda puede utilizar esta señal de manera programada como un prestablecimiento o borrado asincrónico. Por último, dos o cuatro terminales (dependiendo del dispositivo) pueden emplearse como "controles de tres estados globales" (GTS, global three-state controls); una de estas señales puede ser seleccionada en cada macrocelda para la habilitación de salida del manejador de salida correspondiente cuando la salida de la macrocelda se encuentre conectada a una terminal de E/S externa.

Sólo cuatro FB se muestran en la figura, pero la arquitectura XC9500 se escala para acomodar 16 FB en el XC95288. Sin tener en cuenta el miembro de la familia

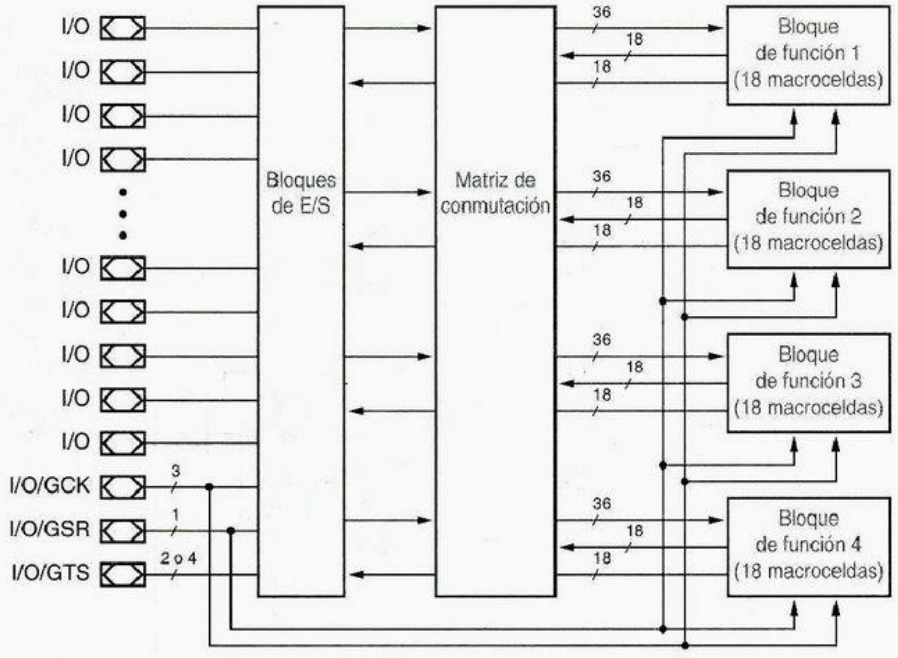


Figura 10-38
Arquitectura de los CPLD de la familia 9500 de Xilinx.

específica, cada FB programable recibe 36 señales desde la matriz de conmutación. Las entradas a la matriz de conmutación son las 18 salidas de macrocelda de cada uno de los FB y las entradas externas de las terminales E/S. Diremos más acerca de cómo la matriz de conmutación se conecta con las cosas en la sección 10.5.4.

Cada FB tiene también 18 salidas que corren "por debajo" de la matriz de conmutación como se dibuja en la figura 10-38 y se conecta a los bloques de E/S. Éstas son meramente las señales de habilitación de salida para los excitadores de salida del bloque de E/S; son utilizados cuando la salida de la macrocelda FB se conecta a una terminal E/S externa.

10.5.2 Arquitectura de bloque de función

La estructura básica de un FB XC9500 se muestra en la figura 10-39. El arreglo programable AND tiene sólo 90 términos de producto. En comparación con los PDL tipo 16V8 y 22V10, el XC9500 y la mayoría de las macroceldas CPLD tienen menos términos AND por macrocelda: donde el 16V8 tiene 8 y el 22V10 tiene 8-16, el XC9500 tiene sólo 5. Sin embargo, no todo es tan malo, debido a la *asignación de término de producto*. El XC9500 y otros CPLD tienen *asignadores de términos de producto* que permiten que

asignación de término de producto

asignador de término de producto

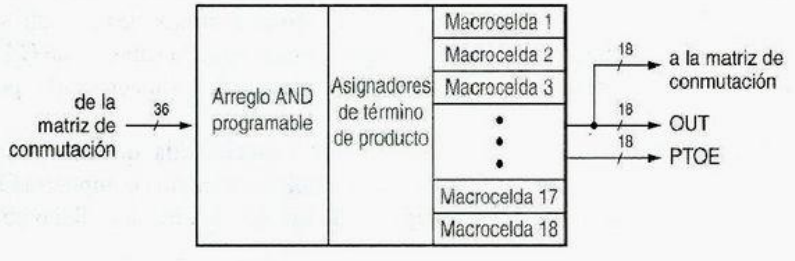


Figura 10-39
Arquitectura de bloque de función (FB, function block).

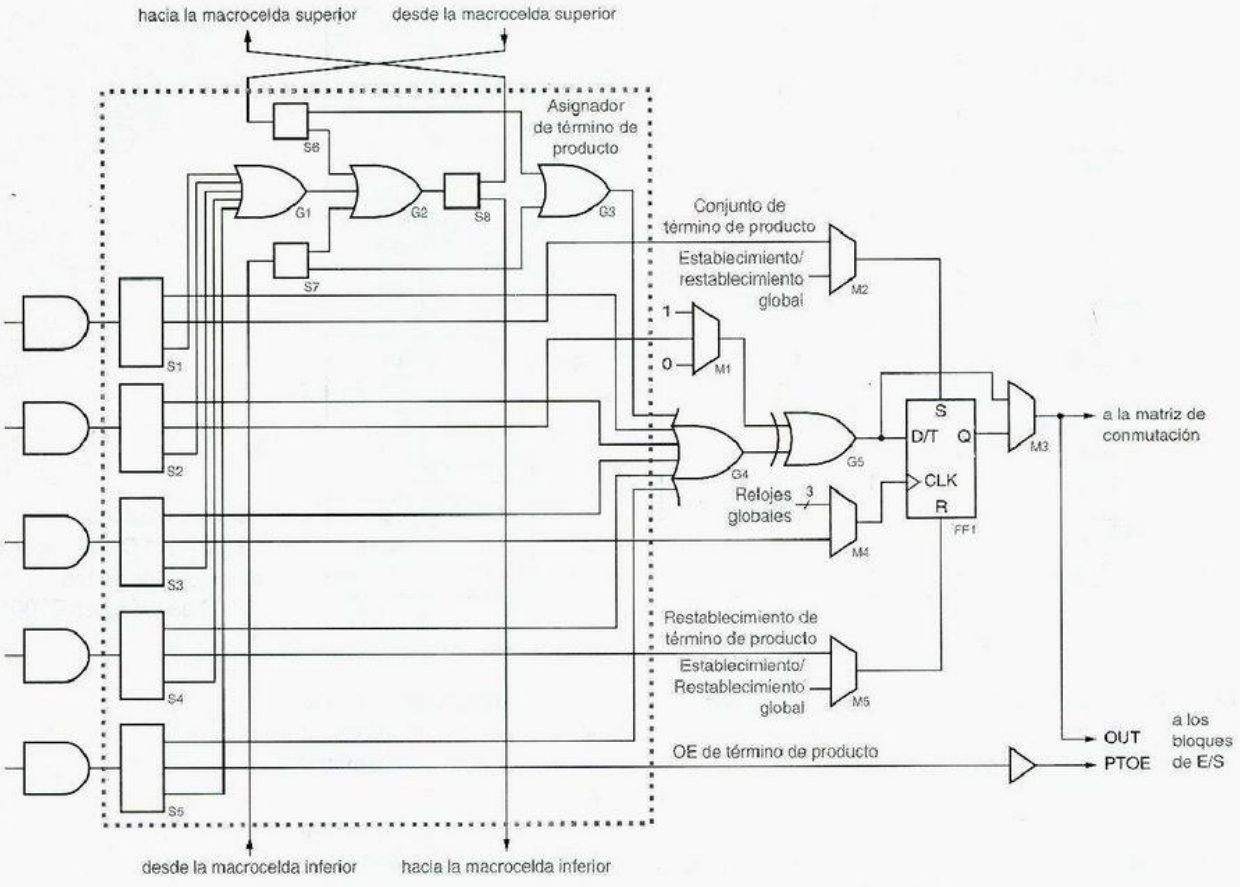


Figura 10-40 Asignador de término de producto y macrocelda XC9500.

términos de producto sin utilizar de una macrocelda sean empleados por otras macroceldas cercanas en el mismo FB.

La figura 10-40 es un diagrama lógico del asignador de términos de producto y macrocelda XC9500. En esta figura, los cuadros rectangulares etiquetados S1-S8 son elementos de dirección de señal programables que conectan sus entradas a una de sus dos o tres salidas. Los cuadros trapezoidales marcados M1–M5 son multiplexores programables que conectan una de sus dos a cuatro entradas a sus salidas.

Las cinco compuertas AND asociadas con la macrocelda aparecen en el lado izquierdo de la figura. Cada una se encuentra conectada a un cuadro de dirección de señal cuya salida superior conecta el término de producto a la compuerta G4 OR principal de la macrocelda. Considerando sólo esto, nada más cinco términos de producto se encuentran disponibles por macrocelda. Sin embargo, la sexta entrada superior de G4 conecta a otra compuerta G3 OR que recibe términos de producto de las macroceldas por arriba y por debajo de la actual.

Cualquiera de los términos de producto de la macrocelda, que no se usan de otro modo, se pueden dirigir a través S1-S5 para combinarlos en una compuerta OR (designada como G1), cuya salida puede dirigirse a la macrocelda hacia arriba o abajo con la

UN CAMINO En una macrocelda XC9500 dada, usted normalmente no dirigiría los términos de producto encadenados de regreso en la dirección en la que vienen. Por ejemplo, si los términos de producto están llegando a S6 desde arriba, podemos utilizarlos localmente al haber S6 dirigirlos a G3, o S6 puede dirigirlos hacia G2. En este último caso, S8 dirigiría la salida de G2 hacia abajo a la macrocelda inferior; no tiene sentido dirigir los términos de producto de vuelta a la macrocelda superior. De hecho, si S6 y S8 en esta macrocelda fueran términos de producto dirigidos hacia arriba, y S7 y S8 la macrocelda superior estuviera dirigiendo los términos de producto hacia abajo, tendríamos un desagradable círculo vicioso.

ayuda de S8. Antes de dirigir, estos términos de producto se pueden combinar con los términos de producto que provienen de abajo o arriba a través de S6, S7 y G2. De este modo, los términos de producto se pueden "encadenar" a través de macroceldas sucesivas para crear sumas de productos más grandes. En principio, se podían combinar y dirigir los 90 términos de producto del FB a una macrocelda, aunque eso dejaría 17 de las 18 macroceldas del FB sin términos de producto.

Además de privar a otras macroceldas de términos de producto, los términos encadenados tienen un precio adicional. Se incurre en un pequeño retardo adicional por cada "salto" que generan los términos de producto dirigidos. Este retardo puede ser minimizado mediante una asignación cuidadosa de macroceldas hambrientas de términos de producto de modo que se encuentren adyacentes a las macroceldas con bajos requerimientos de términos de producto. Por ejemplo, una macrocelda puede utilizar 13 términos de producto con sólo un retardo de salto extra si se encuentra posicionada entre dos macroceldas que utilicen únicamente un término de producto cada una.

La tercera selección media para cada uno de los cuadros de dirección S1-S5 es utilizar el término de producto para una "función especial". Las funciones especiales son establecimiento, restablecimiento y reloj de fuente; control XOR y habilitación de salida. Por lo normal, la mayoría de estas funciones especiales no se usan.

Examinando más de cerca el corazón de la macrocelda, la compuerta G4 OR forma una expresión de suma de productos que utiliza todos los términos de producto seleccionados y alimentándolos en la compuerta G5 XOR. La otra entrada de G5 puede ser 0, 1 o un término de producto, como se seleccione por el multiplexor M1. Estableciendo esta entrada a 1 invierte la expresión de suma de productos de G4, de modo que la macrocelda puede ser configurada para utilizar cualquier polaridad de ecuaciones de lógica minimizada. En el diseño de contadores el hecho de establecer esta entrada a un término de producto es de mucha utilidad. El término de producto se acomoda para que sea 1 cuando los bits de contador de orden inferior son 1 y el conteo está habilitado, y la salida de G4 se arregla para que sea el valor actual del bit contador; de esta manera, el bit contador se complementa como se requiere.

El flip-flop FF1 de la macrocelda puede ser programado para comportarse como si fuera un flip-flop D o un flip-flop T con habilitación; este último también es útil en algunos estilos de realización de contador. El multiplexor M4 selecciona la entrada del reloj del flip-flop de una a cuatro fuentes: las tres entradas del reloj global del CPLD o un término de producto. Esta última elección se prohíbe en las metodologías de diseño sincrónico, excepto en las aplicaciones de sincronización bien definidas, como en la figura 8-111.

El flip-flop también tiene entradas de establecimiento y restablecimiento sincrónicas con fuentes de entrada controladas por los multiplexores M2 y M5. En la mayoría de las aplicaciones, el establecimiento o restablecimiento estaría conectado a la entrada de establecimiento/restablecimiento global del CPLD y se utilizaría sólo en la inicialización del sistema. Sin embargo, estas entradas también se pueden utilizar para tener acceso a un latch S-R en el cual la entrada CLK no sea usada, o, si se es cuidadoso, se puede emplear CLK y S o R en aplicaciones de sincronización como en la figura 8-111.

Un multiplexor final M3 selecciona ya sea la salida del flip-flop o su entrada de datos para utilizarse como la salida de la macrocelda, OUT. Esta señal se envía hacia la matriz de conmutación, donde puede utilizarse por cualquier otra macrocelda. También se envía a los bloques de E/S, junto con un término de producto seleccionado por S5 que puede emplearse como la señal de habilitación de salida PTOE si es necesario.

10.5.3 Arquitectura de bloque de entrada/salida

bloque de E/S (IOB, I/O block)

La estructura del *bloque de E/S (IOB, I/O block)* XC9500 se muestra en la figura 10-41. Existen siete, cuéntelas, siete elecciones de señales de habilitación de salida para el separador del controlador de tres estados. Puede encontrarse siempre encendido, siempre apagado, controlado por el término de producto PTOE de la macrocelda correspondiente, o controlado por cualquiera de hasta cuatro habilitaciones de salida global. Las habilitaciones de salida global son seleccionables como versiones de estado activo alto o estado activo bajo de las terminales GTS externas.

El IOB del XC9500 es un buen ejemplo de una importante tendencia en las arquitecturas de E/S en CPLD y FPGA: proporcionar muchos controles "analógicos" además de los "lógicos" como las habilitaciones de salida. Se suministran tres diferentes controles analógicos:

- *Control de velocidad de sesgo.* El tiempo de ascenso y caída de las señales de salida pueden establecerse para que sean rápidas o lentas. La configuración rápida proporciona el retardo de propagación más rápido posible, mientras que la configuración lenta ayuda a controlar el zumbido en la línea de transmisión y ruido del sistema a expensas de un pequeño retardo adicional.
- *Resistor de arranque.* Cuando se habilita, el resistor de arranque evita que las terminales de salida queden flotando cuando se energiza el CPLD. Esto es útil si las salidas se emplean para manejar las entradas de habilitación de estado activo con otra lógica en que no se supone estarían habilitadas durante el encendido.
- *Tierra programable por el usuario.* Esta característica en realidad reasigna una terminal de E/S como una terminal de tierra, no una terminal de señal en absoluto. Esto es útil en aplicaciones de alta velocidad y tasa de sesgo alta. Las terminales de tierra extra son necesarias para manejar las altas corrientes dinámicas que fluyen cuando múltiples salidas se conmutan de manera simultánea.

Además de estas características, la familia XC9500 proporciona compatibilidad tanto con dispositivos externos de 5 V como de 3.3 V. El separador de entrada y la lógica interna funcionan a partir de una fuente de alimentación de 5 V (V_{CCINT}). Dependiendo del voltaje de operación de los dispositivos externos, el excitador de salida utiliza ya sea una alimentación de 5 V o 3.3 V (V_{CCIO}). Nótese que el resistor de arranque extrae el voltaje de alimentación de E/S, V_{CCIO}. Los diodos *D1* y *D2* se utilizan para fijar voltajes arriba de V_{CCINT} o por debajo de tierra que pueden presentarse debido al zumbido de la línea de transmisión.

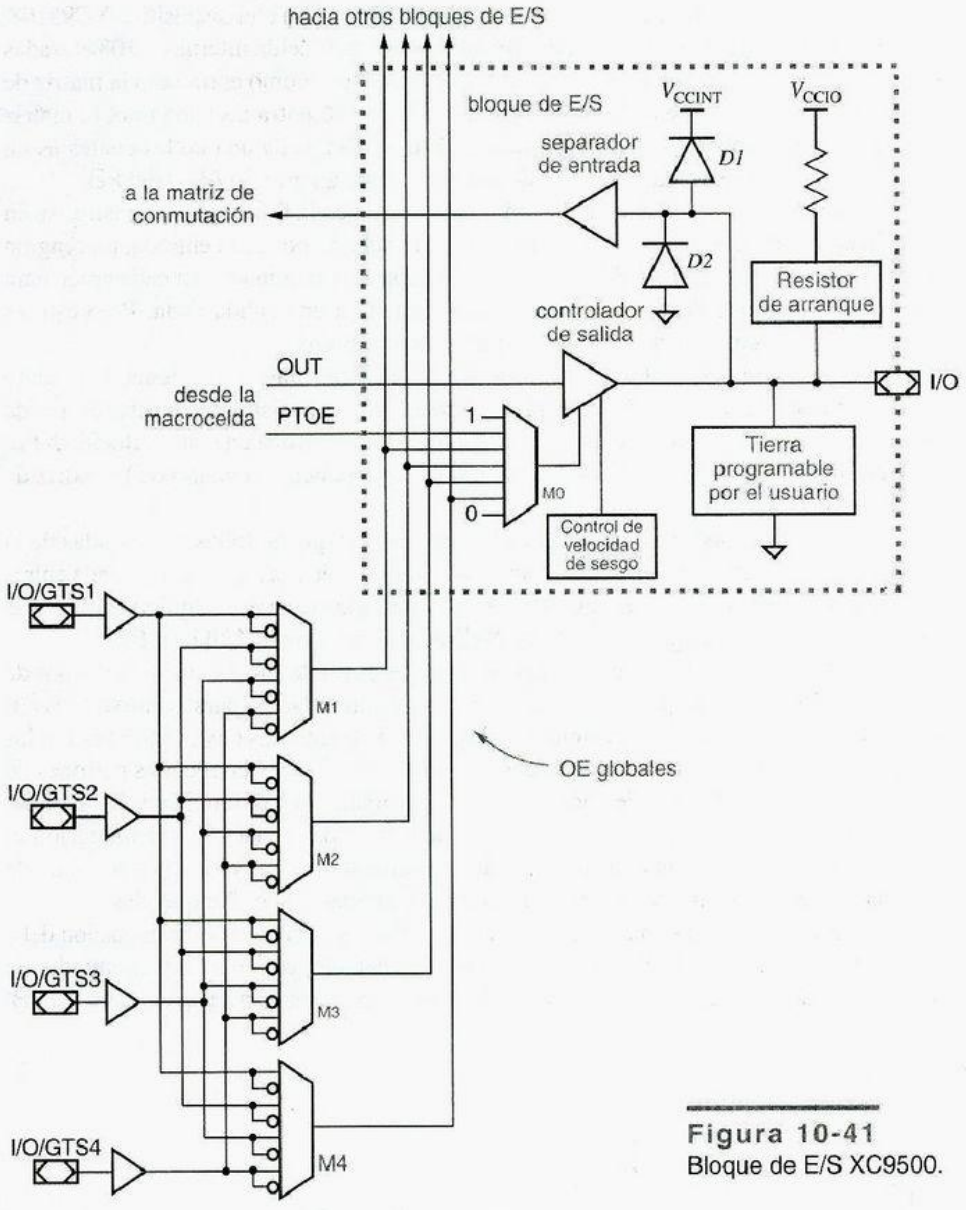


Figura 10-41
Bloque de E/S XC9500.

10.5.4 Matriz de conmutación

En teoría, una interconexión programable de CPLD debería permitir que cualquier salida PLD interna o terminal de entrada externa fuera conectada a cualquier entrada PLD interna. Del mismo modo, debería permitir que cualquier salida PLD se conectara a cualquier terminal de salida externa. Pero si se reflexiona sobre esto, se observará que estamos de vuelta con el mismo problema n^2 que teníamos en la construcción de un 128V64.

El caso de un miembro típico de la familia XC9500 Xilinx, el dispositivo XC95108, se muestra en la figura 10-42. Existen 108 salidas de macrocelda internas y 108 entradas de terminal externa, un total de 216 señales que conectarse como entradas a la matriz de conmutación. Puesto que el XC95108 tiene 6 FB con 36 entradas cada uno, la matriz de conmutación debería tener (casualmente) 216 salidas, cada una de las cuales es un multiplexor de 216 entradas controlando una entrada de un arreglo AND del FB.

Una matriz de conmutación tal como la mostrada en la figura puede construirse en un chip como una estructura rectangular, con una columna por cada entrada, un renglón por cada salida, y un transistor de paso (o compuerta de transmisión) en cada cruce para controlar si una entrada dada se encuentra conectada a una salida dada. Pero esto es todavía una estructura grande: 216 renglones y 216 columnas en el ejemplo.

Con la tecnología de CI de alta densidad de nuestros días, el problema no es tanto el tamaño como la velocidad. Tener un gran número de transistores conectados a cada renglón o columna provoca una alta capacitancia, lo que provoca una baja velocidad. Por lo tanto, los fabricantes de CPLD buscan maneras de reducir el tamaño de la matriz de conmutación.

Por ejemplo, podemos observar en la figura 10-42 que no todas las entradas de la matriz de conmutación deben conectarse a cada salida. Sólo necesitamos que cada entrada pueda conectarse a *alguna* entrada de cada FB, puesto que cualquier entrada FB puede conectarse a cualquier compuerta AND dentro del arreglo AND del FB.

Entonces de nueva cuenta, el requerimiento no es así de simple como acabamos de establecer. Supongamos que la salida de la matriz de conmutación para la entrada FB i ($0 \leq i \leq 35$) es creada mediante un simple multiplexor de 8 entradas cuyas entradas son las entradas de la matriz de conmutación desde $8i$ hasta $8i + 7$. Existen muchos patrones de interconexión que no se pueden acomodar con una matriz de conmutación de este tipo. Por ejemplo, no sería posible conectar las entradas 0-35 de la matriz de conmutación al mismo FB. Tan pronto como la entrada 0 de la matriz de conmutación es conectada a la entrada 0 de FB, las entradas 1-7 de la matriz de conmutación son bloqueadas.

De esta manera, el requerimiento de conectividad de la matriz de conmutación debe ser establecido con mayor amplitud: para cada FB, cualquier combinación de entradas de matriz de conmutación debe poder conectarse en alguna combinación de las entradas del FB.

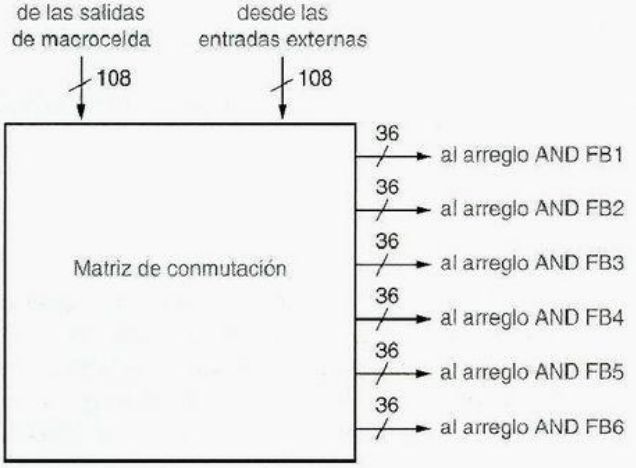


Figura 10-42
Requerimientos de la matriz de conmutación XC95108.

Una típica matriz de conmutación CPLD es un compromiso entre el esquema de multiplexor mínimo y un arreglo completo de puntos de cruce, sin bloqueo. Con algo menos que un arreglo de puntos de cruce sin bloqueo, el problema de la asignación de conexiones de entrada-salida en la matriz de conmutación es no trivial. Para cada diferente diseño basado en CPLD, debe encontrarse un conjunto de conexiones de matriz de conmutación mediante software de "ajuste" proporcionado al diseñador por el fabricante de CPLD.

Hallar un conjunto completo de conexiones a través de una matriz de conmutación dispersa es uno de esos problemas *NP*-completos que usted habrá escuchado en las ciencias de la computación. En términos legibles para principiantes, esto significa que para algunos diseños, el software de ajuste puede tener que ejecutarse mucho más tiempo del que a usted le gustaría esperar para averiguar si existe una solución. Si la matriz de conmutación tiene muy pocos puntos de cruce, como en nuestro ejemplo del multiplexor mínimo, incluso la ejecución indefinida del mejor software de ajuste no será suficiente para hallar un conjunto completo de conexiones para algunos diseños.

De esta forma, el diseño de una matriz de conmutación de CPLD es un compromiso entre el desempeño del chip (velocidad, área, costo) y las capacidades del software de ajuste. Este software de ajuste por lo regular determina no sólo las conexiones finales a través de la matriz de conmutación, sino también la asignación de las entradas y salidas del CPLD a FB, macroceldas y terminales externas, y de "lógica enterrada" hacia FB y macroceldas. Estas asignaciones interactúan a su vez con ambos problemas de conexión de matriz de conmutación y de asignación de términos de producto. Las soluciones a estos problemas son el "ingrediente secreto" del diseño de software y chip CPLD, y habitualmente no es revelado por los fabricantes de los CPLD.

ASEGURAMIENTO DE TERMINAL

Otra cuestión importante en el diseño de software y chip CPLD es el *aseguramiento de terminal*. En la mayoría de las aplicaciones CPLD, está bien dejar que el software de ajuste seleccione cualquier terminal que desee para las señales de entrada y salida externas del dispositivo. Sin embargo, una vez que el diseño está completo y que se ha fabricado un PCB, el diseñador desearía "asegurar" las asignaciones de terminales de modo que permanezcan siendo siempre las mismas incluso si se hicieran pequeños (¡o grandes!) cambios al diseño para eliminar errores. Esto permite ahorrar tiempo, costo y evitar las molestias de volver a trabajar o rediseñar y volver a fabricar el PCB.

Las asignaciones de terminal "asegurada" son habitualmente especificadas en un archivo que se lee mediante el software de ajuste. Con los primeros CPLD y FPGA, asegurar las terminales antes de hacer un pequeño cambio no garantizaría el éxito: el software de ajuste podría renunciar y quejarse de estar demasiado reprimido. Si se liberan las terminales, el software de ajuste podría hallar una nueva localidad que funcionara, pero sería completamente cambiado del original.

Estos problemas no serían necesariamente una falla del software de ajuste; los CPLD y FPGA simplemente no tienen suficiente conectividad interna para respaldar frecuentes cambios de diseño bajo la limitante del aseguramiento de terminales. Los fabricantes de dispositivos han aprendido de esta experiencia y han mejorado sus arquitecturas internas de dispositivos para acomodar frecuentes cambios de diseño. Por ejemplo, algunos dispositivos contienen una "matriz de conmutación de salida" que garantiza que cualquier señal interna de macrocelda pueda ser conectada a cualquier terminal de E/S externa.

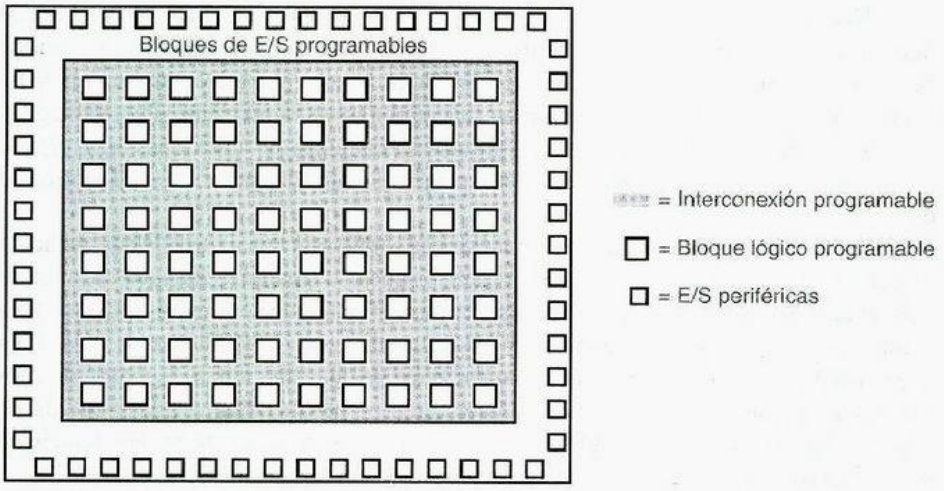


Figura 10-43
Arquitectura general del chip FPGA.

10.6 Arreglos de compuertas programable por campo

Un arreglo de compuertas programable por campo (FPGA, field-programmable gate array), es algo parecido a un CPLD volteado, puesto de adentro hacia fuera. Como se muestra en la figura 10-43, la lógica se divide en un gran número de bloques lógicos programables que son individualmente más pequeños que un PLD. Se encuentran distribuidos a través de todo el chip en un mar de interconexiones programables y todo el arreglo se encuentra rodeado de bloques de E/S programables. Un bloque lógico programable de FPGA es menos eficiente que un PLD típico, pero un chip FPGA contiene mucho más bloques lógicos que los PLD que contiene un CPLD del mismo tamaño de dado.

Xilinx, Inc. inventó los FPGA, y en esta sección utilizaremos una de sus populares familias, la XC4000E, para ilustrar la arquitectura FPGA.

10.6.1 La familia FPGA XC4000 de Xilinx

bloque lógico configurable (CLB, configurable logic block)

Los bloques lógicos programables en la familia de FPGA XC4000E de Xilinx se denominan *bloques lógicos configurables (CLB, configurable logic blocks)*. La parte más pequeña, a XC3003E, contiene un arreglo de 10×10 de CLB, mientras que la mayor, la XC4025E, contiene un arreglo 32×32 para un total de 1,024 CLB. Xilinx también ha creado las familias extendidas XC4000EX y XC4000XL, basadas en la familia XC4000E, que tienen recursos y características adicionales que no se discuten aquí. El miembro más grande de las familias extendidas, el XC4085XL, tiene 3,136 CLB. La tabla 10-9 de Xilinx muestra los miembros de la familia que se encontraban disponibles en 1999.

Como parte de una familia CPLD, la familia XC4000 abarca una amplia gama de tamaños de dispositivo y capacidades de entrada/salida. La columna de la tabla que está marcada como "Max. User I/O" hace referencia al número máximo de bloques de entrada/salida que se proporcionan en el chip. Sin embargo, los dispositivos XC4000 están disponibles en una amplia gama de encapsulados, pero no todas las E/S (I/O) se conectan a las terminales externas de los encapsulados más pequeños. Como sucede con los CPLD, el usuario de FPGA tiene la oportunidad de escalar un diseño desde los dispositivos más pequeños hasta los más grandes en el mismo encapsulado, o de un encapsulado más pequeño a uno más grande.

Tabla 10-9 Recursos en los FPGA serie XC4000 de Xilinx.

Dispositivo	*Matriz CLB*	*CLB totales*	*Max. E/S del usuario*	*Flip-flops*	*Max. bits de RAM (sin lógica)*	*Max. compuertas (sin RAM)*	*Típico rango de compuerta (lógica y RAM)*
XC4002XL	8×8	64	64	256	2,048	1,600	1,000–3,000
XC4003E	10×10	100	80	360	3,200	3,000	2,000–5,000
XC4005E/XL	14×14	196	112	616	6,272	5,000	3,000–9,000
XC4006E	16×16	256	128	768	8,192	6,000	4,000–12,000
XC4008E	18×18	324	144	936	10,368	8,000	7,000–15,000
XC4010E/XL	20×20	400	160	1,120	12,800	10,000	7,000–20,000
XC4013E/XL	24×24	576	192	1,536	18,432	13,000	10,000–30,000
XC4020E/XL	28×28	784	224	2,016	25,088	20,000	13,000–40,000
XC4025E	32×32	1,024	256	2,560	32,768	25,000	15,000–45,000
XC4028EX/XL	32×32	1,024	256	2,560	32,768	28,000	18,000–50,000
XC4036EX/XL	36×36	1,296	288	3,168	41,472	36,000	22,000–65,000
XC4044XL	40×40	1,600	320	3,840	51,200	44,000	27,000–80,000
XC4052XL	44×44	1,936	352	4,576	61,952	52,000	33,000–100,000
XC4062XL	48×48	2,304	384	5,376	73,728	62,000	40,000–130,000
XC4085XL	56×56	3,136	448	7,168	100,352	85,000	55,000–180,000

La columna "Flip-Flops" cuenta todos los flip-flops en el dispositivo, dos por CLB y dos por bloque de E/S, como veremos más adelante. Solamente una fracción de los flip-flops disponibles se utiliza en un diseño típico, pero este número es una cifra de interés que los diseñadores buscan cuando aplican un FPGA un diseño. La columna "Max. bits de RAM" es otra cifra de interés. Como veremos, en lugar de ser utilizada para lógica, cada CLB puede ser configurado como una pequeña SRAM almacenando hasta 32 bits.

El número de "Max. compuertas" es difuso. Para el XC4002XL, la tabla afirma que cada CLB puede realizar la misma función que aproximadamente 25 compuertas en un diseño discreto. El XC4003E es aún mejor, con 30 compuertas por CLB. ¿Esto es razonable? ¿Y qué es con exactitud una "compuerta"? ¿Las compuertas XOR o NAND amplias cuentan como una compuerta? Podrá decidir por usted mismo, después de que conozca más acerca de la arquitectura CLB. Al mismo tiempo, decidirá si esta columna fue creada por personas del área de ingeniería o de mercadotecnia.

La última columna de la tabla en verdad ha sido escrita por personal de mercadotecnia, puesto que el extremo superior de cada compuerta "típica" ¡es mayor que el máximo en la columna antes de ella! En realidad, *existe* una explicación razonable para esta aparente inconsistencia. Esta columna supone que el 20%-30% de los CLB se utilizan para memoria más que para lógica, con 32 bits de SRAM por CLB. La mínima implementación a nivel de compuerta de una celda SRAM es un latch D empleando cuatro

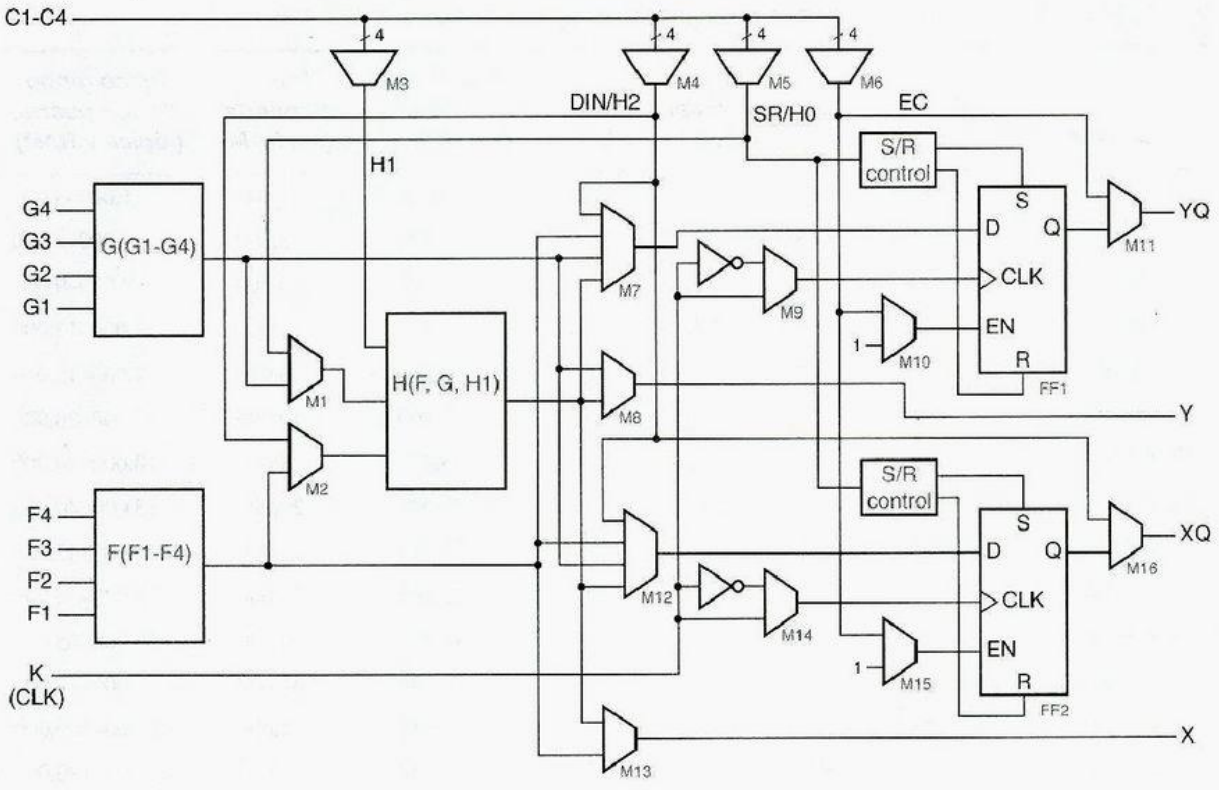


Figura 10-44 Bloque lógico configurable XC4000.

compuertas (figura X7.27 en la página 650), de modo que podemos contar cada CLB como 128 compuertas cuando se utiliza como SRAM. El número en la última columna, por consiguiente, es el de las compuertas utilizadas para la lógica *más* el de compuertas equivalentes empleadas para SRAM.

10.6.2 Bloque de lógica configurable

Puesto que un FPGA puede tener grandes cantidades de CLB, ¡es importante que entendamos éstos en primer lugar! La figura 10-44 muestra la estructura interna de un CLB serie XC4000.

Los elementos programables más importantes de CLB son los generadores de función lógica F, G y H. Tanto F como G pueden realizar cualquier función lógica combinacional de sus cuatro entradas, y H puede efectuar cualquier función lógica combinacional de sus tres entradas.

¿Cómo funcionan F, G y H? Dada la tarea de construir un generador de función "universal" para funciones lógicas de 4 entradas, ¿qué clase de circuito al nivel de compuerta podría idearse? Piense acerca de ello, y regresaremos al tema más adelante.

Como en nuestra discusión sobre CPLD, los cuadros con forma de trapecio de la figura 10-44 representan multiplexores programables. Nótese que las salidas de F y G,

además de las entradas adicionales CLB, se pueden enviar a las entradas de H mediante los multiplexores M1-M3, para facilitar la realización de algunas funciones con más de cuatro entradas. A continuación se ofrece una taxonomía de las funciones que se pueden realizar mediante F, G y H en un CLB simple:

- Cualquier función de hasta cuatro variables, más cualquier segunda función de hasta cuatro variables no relacionadas, más cualquier tercera función de hasta tres variables no relacionadas.
- Cualquier función simple de cinco variables (véase el ejercicio 10.34).
- Cualquier función de cuatro variables, más algunas segundas funciones de seis variables no relacionadas.
- Algunas funciones de hasta nueve variables, incluyendo verificación de paridad y un verificador de igualdad en cascada para dos entradas de 4 bits (véase ejercicios 10.35 y 10.36).

Con la apropiada programación de los multiplexores M7-M8 y M12-M13, las salidas de los generadores de función pueden dirigirse a las salidas X y Y del CLB, o se pueden capturar en los flip-flops D que se disparan por los flancos FF1 y FF2. Los flip-flops pueden utilizar el flanco de ascenso o de caída de una entrada del reloj común, K, como la que seleccionan los multiplexores M9 y M14. También pueden utilizar una señal de habilitación del reloj, EC, seleccionada por M10 y M15. Las fuentes de las tres señales internas y las señales EC se seleccionan de un conjunto de cuatro entradas especiales C1-C4 mediante los multiplexores M3-M6 en la parte superior del CLB.

Las salidas XQ y YQ del CLB llevan las salidas del flip-flop fuera del CLB. Si un flip-flop no es utilizado en el CLB, el multiplexor M11 o el M16 pueden seleccionar XQ o YQ como una "salida de desviación" que simplemente es una copia de una entrada de CLB seleccionada por M4 o M6.

El bloque que está identificado como "S/R control" de cada flip-flop determina si éste se encuentra establecido o restablecido en la configuración. También determina si el flip-flop responde a una señal de establecimiento/restablecimiento global (que no se muestra) o si responde a la señal SR del CLB que seleccionó el multiplexor M5.

¡Qué abundancia de programabilidad! Por supuesto, la configuración de CLB dentro de una parte XC4000, si tiene 3,136 CLB o sólo 64, no es llevada a cabo en forma manual. El fabricante proporciona una herramienta de ajuste que asigna, configura y conecta los CLB para que satisfagan una descripción de diseño de alto nivel escrita en ABEL, VHDL Verilog o de forma esquemática.

Regresemos a nuestra pregunta de cómo construir un generador de función universal para funciones lógicas de 4 entradas. Es un problema difícil si se piensa de él al nivel de compuerta, pero bastante fácil si se le piensa desde otro punto de vista. Cualquier función lógica de 4 entradas puede describirse mediante su tabla de verdad, la cual tiene 16 renglones. Supongamos que almacenamos ésta en una memoria de 16 palabras por 1 bit de amplitud. Cuando aplicamos los bits de entrada de la función a las líneas de dirección de la memoria, su salida de datos es el valor de la función para esa combinación de entrada.

Éste es con exactitud el enfoque que fue tomado por los diseñadores de FPGA en Xilinx. Los generadores de función F y G son en realidad unas SRAM de 16 × 1 muy compactas y rápidas, y H es una SRAM de 8 × 1. Cuando un CLB se emplea para realizar lógica, las tablas de verdad de las funciones lógicas F, G y H son cargadas en SRAM en tiempo de configuración desde una memoria sólo de lectura externa. Los multiplexores programables de la figura 10-44 también se controlan mediante una "SRAM", de hecho, los latchs D individuales también se cargan en el tiempo de configuración. Esta programación se hace para todos los CLB en el FPGA.

Aparte de la conveniencia, el uso de la memoria para almacenar las tablas de verdad tiene otro beneficio importante. Cualquier CLB XC4000 puede configurarse durante el arranque para usarlo como memoria en lugar de lógica. Se pueden configurar varios modos diferentes:

- *Dos SRAM de 16 × 1.* F y G son utilizadas como SRAM con direcciones independientes y entradas de escritura de datos. Sin embargo, comparten una entrada de habilitación de escritura común.
- *Una SRAM de 32 × 1.* Los mismos cuatro bits de dirección son empleados para F y G, y un quinto bit de dirección es aplicado al generador de función H y los circuitos de habilitación de escritura para seleccionar entre F y G, las mitades superior e inferior de la memoria.
- *Asincrónica o sincrónica.* Para operaciones de escritura, las SRAM anteriores pueden configurarse para obtener un comportamiento de latch asincrónico "normal" o pueden configurarse para tener lugar en un flanco designado de la señal del reloj K.
- *Una SRAM de puerto dual de 16 × 1.* Los dos conjuntos de entradas de dirección se emplean para leer y escribir en forma independiente diferentes localidades en la misma SRAM. Únicamente las operaciones de escritura sincrónicas son respaldadas en este modo.

En estos modos, las entradas de función F1-F4 y G1-G4 suministran direcciones, otras entradas CLB H0-H2 proporcionan las entradas de datos y la señal de habilitación de escritura y las salidas de datos se proporcionan en las salidas de generador F y G y pueden capturarse en los flip-flops FF1 y FF2 o salir en las salidas X y Y del CLB.

10.6.3 Bloque de entrada/salida

Bloque de E/S (IOB, I/O block)

La estructura del *bloque de E/S (IOB, I/O block)* XC4000 se muestra en la figura 10-45. Una terminal de E/S puede utilizarse como entrada o salida, o ambas.

El IOB XC4000 tiene más controles "lógicos" que su primo en el CPLD XC9500. En particular, sus trayectorias de entrada y salida contienen flip-flops D disparados por flanco, los cuales se seleccionan mediante los multiplexores M5-M7. La localidad de los flip-flops de entrada y salida "acercándose" a las terminales E/S del dispositivo es útil sobre todo en FPGA. En la salida, los retardos más o menos largos de las salidas de flip-flop CLB internas a los IOB pueden hacer difícil la conexión a sistemas sincrónicos externos a velocidades de reloj muy altas. En la entrada, los retardos largos de las terminales de

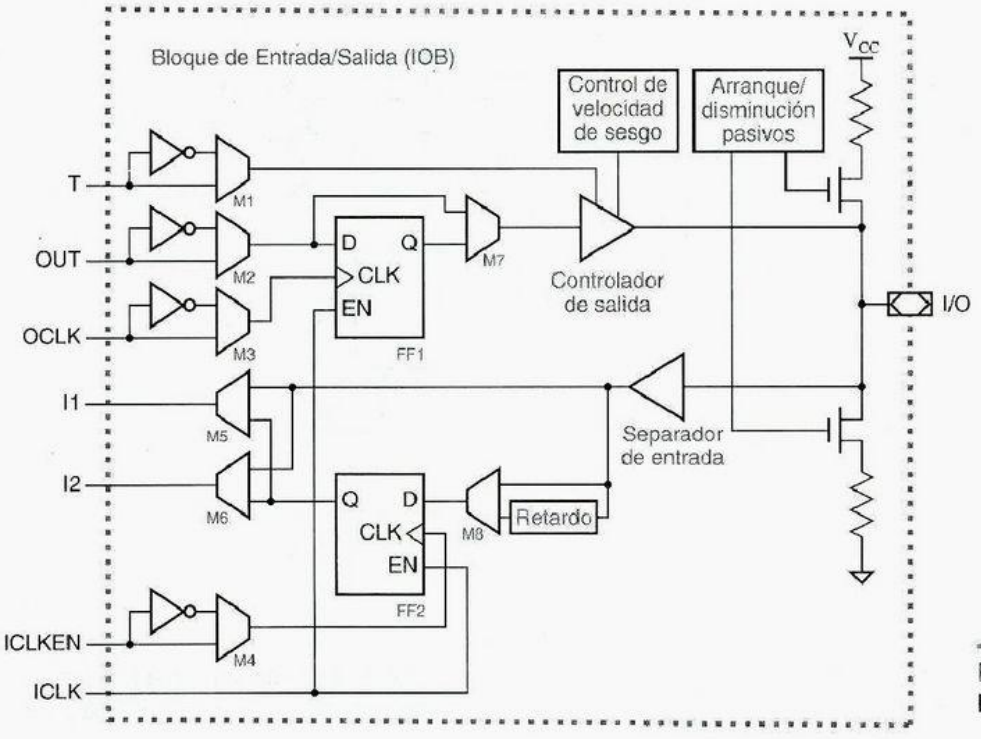


Figura 10-45
Bloque de E/S XC4000.

E/S a las entradas de flip-flop del CLB pueden hacer difícil de satisfacer los tiempos de establecimiento y retención del sistema externo, si las entradas externas están temporizadas directamente en un flip-flop de CLB, sin ser capturadas primero por un flip-flop en la terminal del IOB. Por supuesto, es posible utilizar los flip-flops del IOB sólo si las especificaciones de la interfaz externa de FPGA permiten la "canalización" de entradas y salidas.

Para las entradas canalizadas (pipelined inputs), el IOB XC4000 toma ventaja al proporcionar un elemento de retardo, seleccionable mediante M8, que está en serie con la entrada D del flip-flop FF2 de entrada. El efecto de este elemento es retardar la entrada D con relación a las copias internas del FPGA del reloj de sistema, para garantizar que la entrada tendrá un requerimiento de tiempo de retención igual a cero, con relación al reloj externo del sistema. Este beneficio se obtiene a expensas de un incremento en el tiempo de establecimiento, naturalmente.

Los controles lógicos de otro IOB tienen una polaridad seleccionable, utilizan multiplexores M1-M4 para sus cuatro entradas que vienen desde el arreglo CLB vía la interconexión programable. Estas entradas, OUT, T, OCLK e ICLKEN, son el bit de entrada, su habilitación de tres estados, del reloj de salida y la habilitación del reloj de entrada, respectivamente.

Como el IOB XC9500, el IOB XC4000 también tiene controles analógicos. La velocidad de sesgo del excitador de salida es programable, y se puede conectar un resistor de arranque o disminución en la terminal de E/S.

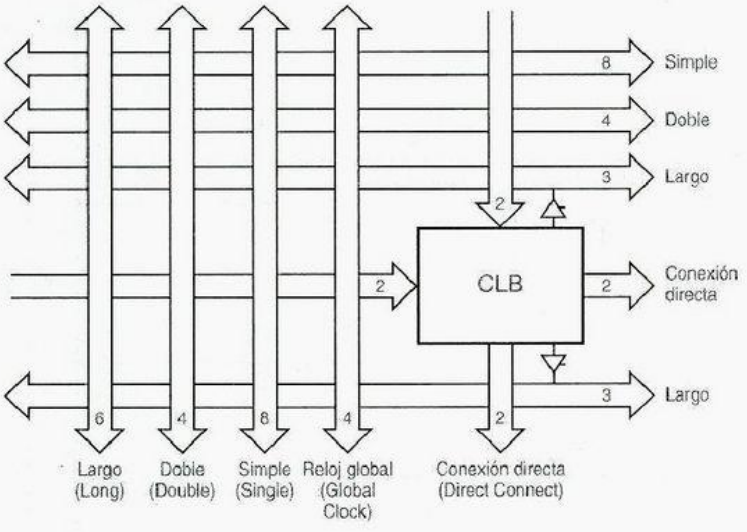


Figura 10-46
Estructura de interconexión general del XC4000.

10.6.4 Interconexión programable

Bien, hemos dejado lo mejor para el final. La arquitectura de interconexión programable del XC4000 es un fascinante ejemplo de una estructura que proporciona una rica conectividad simétrica en una área reducida de silicio.

Como hemos mostrado en la figura 10-43, cada CLB en un FPGA se encuentra incrustado en la estructura de interconexión, que se compone en realidad de alambres con conexiones programables para ellos. La figura 10-46 ofrece un poco más de detalle acerca del esquema de conexión del XC4000. Los alambres no son en realidad "propiedad" de algún CLB, pero un arreglo CLB XC4000 es creado al tapizar el chip con exactamente la estructura ilustrada en la figura. Por ejemplo, 100 copias de esta figura hacen el arreglo CLB de 10×10 de un XC4003.

El número en cada flecha indica el número de alambres en esa trayectoria de señal. De este modo, usted puede ver que un CLB tiene dos alambres (salidas) y cada una llega a los CLB de inmediato por debajo y a la derecha de él. También conecta los tres grupos de alambres arriba, uno debajo y cuatro a su izquierda. Las señales en estos alambres pueden fluir en cualquier dirección.

Los cuatro alambres en el grupo etiquetado "Reloj Global" son optimizados para su uso como entradas de reloj a los CLB, proporcionando un retardo corto y un sesgo mínimo. Los dos grupos "simples" son optimizados para conectividad flexible entre bloques adyacentes, sin el número pequeño y limitación unidireccional de alambres en los grupos "conexión directa".

Es posible conectar un CLB a otro que se encuentra a más de un salto de distancia mediante el uso de alambres "sencillos", pero los alambres tienen que instalarse a través de un conmutador programable para cada salto, lo que agrega retardos adicionales. Los alambres en los grupos "doble" viajan pasados dos CLB antes de llegar a un conmutador,

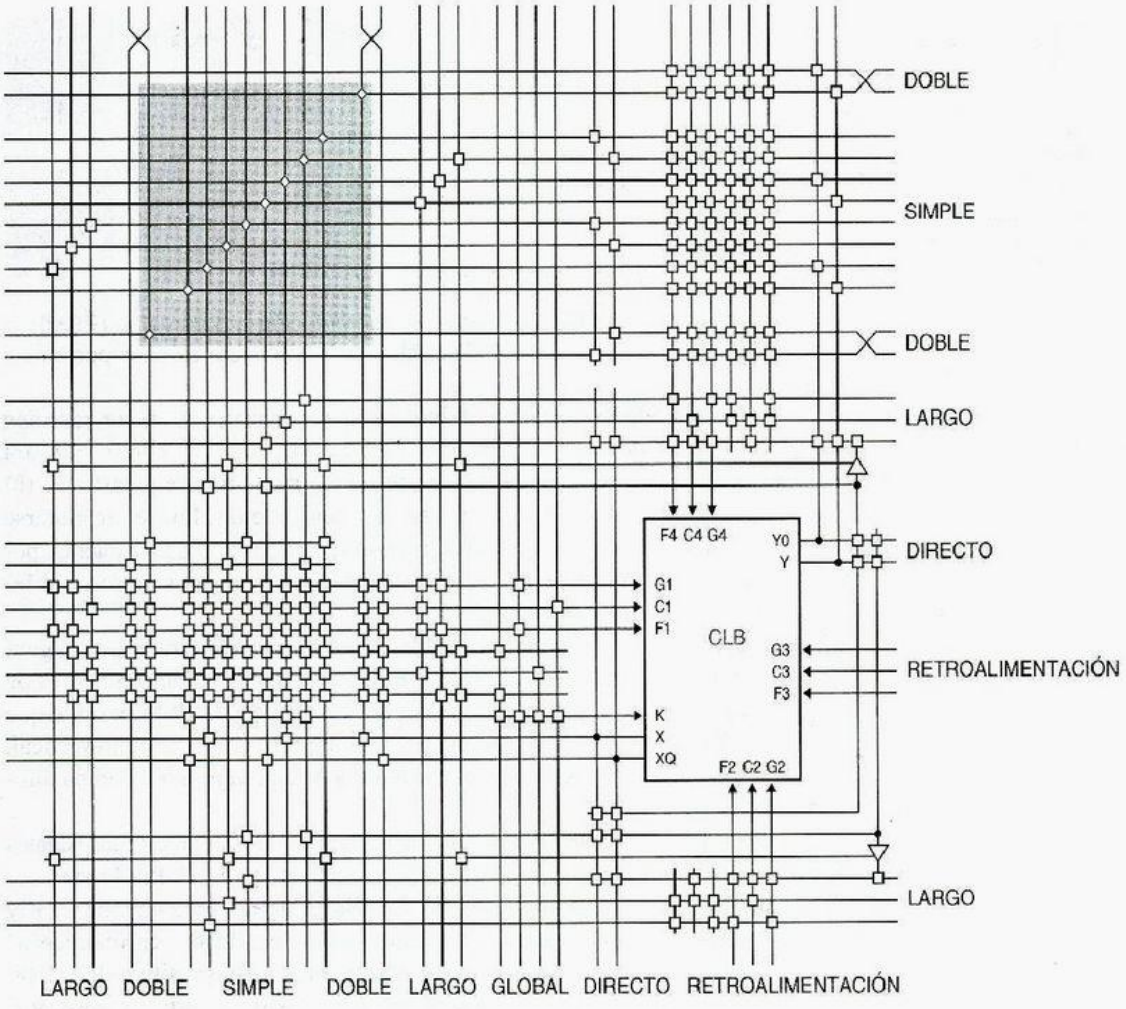


Figura 10-47 Detalles de conexión de alambres y CLB del XC4000.

de modo que proporcionan retardos más cortos para conexiones más largas. Para conexiones realmente largas, los grupos "Largo" no pasan por ningún conmutador programable en absoluto; por el contrario, viajan todo el camino a través o hacia debajo de un renglón o columna y son controlados por excitadores de tres estados cerca del CLB.

La figura 10-47 muestra un CLB y alambres con mucho más detalle. Los cuadrados pequeños son conexiones programables: un alambre horizontal es conectado o no a uno vertical, dependiendo del estado del bit de programación (otra vez, en un latch) para ese conmutador. Se proporciona una interconexión programable especializada adicional para los flancos del arreglo CLB para conexiones a los IOB.

matriz de conmutación programable (PSM)

En la figura, el área sombreada en color se conoce como una *matriz de conmutación programable (PSM, programmable switch matrix)*. La PSM se ilustra con más detalle

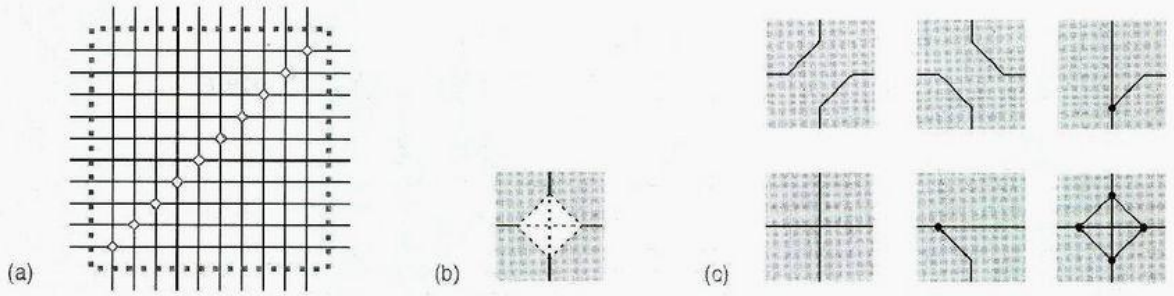


Figura 10-48 Conexiones programables del XC4000: (a) matriz de conmutación programable (PSM); (b) elemento de conmutación programable (PSE); (c) unas cuantas conexiones posibles.

elemento de conmutación programable (PSE)

en la figura 10-48. Cada uno de los diamantes de la figura (a) es un *elemento de conmutación programable (PSE)* que puede conectar cualquier línea a cualquier otra, como se ilustra en (b). Con cuatro líneas, existen 6 posibles pares de conexiones como se muestra en (b) y el PSE tiene una compuerta de transmisión para cada una de ellas. Pueden habilitarse algunas, ninguna o todas las compuertas de transmisión en un PSE; de nueva cuenta, por bits de configuración en latchs. De esta forma, se pueden tener muchos patrones diferentes de conexión, como se muestra en (c).

La PSM es esencial para conectar cosas en la estructura de alambrado de la figura 10-47. Al habilitar y deshabilitar conexiones, los PSE extienden o aíslan segmentos de alambres en los grupos "simple" y "doble". De mayor importancia, la PSM permite que las señales "den vuelta a la esquina" al conectar un alambre horizontal con uno vertical. Sin esto, los CLB no serían capaces de conectarse a otros en un renglón o columna diferentes del arreglo.

Mientras que la PSM es esencial, el hecho de utilizarla tiene su precio: las señales incurren en un pequeño retardo cada vez que saltan a través de un PSE. Por lo tanto, el software de ajuste FPGA de alta calidad examina no sólo cualquier localidad de CLB y conexiones de alambres que funcionen. La herramienta de "localidad y enrutamiento" utiliza mucho tiempo para tratar de optimizar el rendimiento del dispositivo. Para esto tiene que encontrar una localidad que permita el uso de conexiones cortas, y posteriormente debe realizar el enrutado de las mismas conexiones.

Al igual que los CPLD, los FPGA son juzgados por la flexibilidad de sus arquitecturas y la consistencia de los resultados obtenidos de un ajuste después de que se han efectuado pequeños cambios de diseño. No hay nada más frustrante que realizar un pequeño cambio a un diseño extenso y encontrar que ya no satisface los requerimientos de temporización. De este modo, los fabricantes de FPGA han aprendido a proporcionar recursos "extra" en sus arquitecturas para ayudar a asegurar resultados consistentes.

BUENA PRÁCTICA La colocación y enrutamiento es en realidad un problema mal comprendido, porque representa la mayor parte del "extremo terminal" de cualquier diseño de chip personalizado. Así, la misma clase de herramientas y los mismos vendedores de herramientas están involucrados en la colocación y enrutamiento tanto para dispositivos FPGA como para dispositivos ASIC. ¡De este modo, usted deseará aplicar cualquier experiencia que tenga en el diseño de dispositivos FPGA al diseño de dispositivos ASIC!

Referencias

Los fabricantes de ROM y RAM publican libros de datos con sus especificaciones de dispositivo para sus partes, y la mayoría publica hojas de datos individuales y notas de aplicación en sus sitios en la red de Internet. La información de memorias EPROM que se borran con luz ultravioleta puede encontrarse en el sitio de Fairchild Semiconductor, `www.fairchildsemi.com`. El sitio de Advanced Micro Devices, `www.amd.com`, tiene información detallada acerca de los más recientes EPROM y memoria flash.

Los sitios de la red de Internet que contienen toda clase de datos acerca de memorias SRAM y DRAM, antiguas y nuevas, incluyen a Hitachi Semiconductor (`semiconductor.hitachi.com`) y NEC Electrónicos (`www.necel.com`). Las últimas y más grandes SSRAM son ofrecidas por compañías como Integrated Device Technology (`www.idt.com`), Motorola Semiconductor (`www.mot.com`) y Micron Technology (`www.micron.com`).

Las publicaciones de la industria como *Electronic Systems* (`www.estd.com`) y *Electronic Design* son artículos tutoriales y de actualización, con cierta periodicidad, que describen las capacidades y aplicaciones de los más recientes chips de memoria; únicamente vaya a los sitios de la red de Internet y busque "SDRAM" para comenzar. Las nuevas arquitecturas DRAM incluyen la Rambus DRAM (RDRAM), la DRAM sincrónica mejorada (ESDRAM, enhanced synchronous DRAM), SyncLink DRAM (SLDRAM) y la DRAM sincrónica de doble velocidad de datos (DDR-DRAM, double-data-rate DRAM).

Además de las memorias estudiadas en este capítulo, diversos tipos de dispositivos de memoria "de especialidad" tienen un amplio uso. Probablemente las más comunes sean las *memorias FIFO (first-in, first-out, primeras entradas, primeras salidas)*; éstas se utilizan por lo regular para transferir datos de un dominio de reloj o procesador a otro. Los sitios de la red de Internet que corresponden a: IDT, Motorola y Texas Instruments (`www.ti.com`) son buenas fuentes de información acerca de las FIFO. Aunque puede parecer que las FIFO resuelven por arte de magia los problemas de la metaestabilidad que se presentan cuando se cruzan los dominios del reloj, no lo hacen. Por ejemplo, véase "FIFO Memories: Solutions to Reduce FIFO Metastability" por Tom Jackson (Texas Instruments publ. SCAA011A; marzo de 1996).

memorias primero que entra, primero que sale (FIFO, first-in, first-out)

Otra clase de memoria especializada es la de puerto dual, la cual tiene dos conjuntos independientes de líneas de control, datos y dirección, y permiten que sean realizadas operaciones independientes en ambos puertos de manera simultánea. Integrated Device Technology es una fuente líder para estos dispositivos; además de hojas de datos, su sitio en la red de Internet tiene un excelente conjunto de notas de aplicaciones acerca de los dispositivos.

Diversos fabricantes introdujeron diferentes arquitecturas CPLD y FPGA casi al mismo tiempo. Mientras que Xilinx (`www.xilinx.com`) era y es un líder en FPGA, el campo de CPLD es mucho más competido. Las primeras dos familias CPLD fueron presentadas a principios de la década de 1990 por Altera Corporation (`www.altera.com`) y Advanced Micro Devices. La línea de productos CPLD de AMD pertenece ahora a un antiguo competidor Lattice Semiconductor (`www.latticesemi.com`). Cypress Semiconductor (`www.cypress.com`) es otro importante jugador en el Mercado de CPLD. Todas estas compañías tienen excelentes recursos disponibles en sus sitios de la red de Internet.

Problemas propuestos

10.1 Determine el tamaño de la ROM necesaria para realizar la función lógica combinacional en cada una de las siguientes figuras: 4-39(b), 5-39, 5-77, 6-1 y 6-6.

10.2 Determine el tamaño de la ROM necesaria para realizar la función lógica combinacional desempeñada por cada una de las siguientes partes MSI: 74x49, 74x139, 74x153, 74x257, 74x381, 74x682.

10.3 Dibuje un símbolo lógico y determine el tamaño para una ROM que realiza una función lógica combinacional que puede efectuar la función ya sea de un 74x381 o un 74x382 dependiendo del valor de una entrada simple MODE.

10.4 Dibuje un símbolo lógico y determine el tamaño para una ROM que realiza un multiplicador combinacional de 8×8.

10.5 Muestre cómo diseñar una SRAM de $2M \times 8$ utilizando SRAM HM628512 y una parte combinacional MSI como bloques de construcción.

Ejercicios

10.6 Nuestra discusión de trayectorias furtivas de ROM en conexión con la figura 10-6 afirmaba que con A2-A0 = 101, las líneas de bit D2_L y D0_L son llevadas al nivel BAJO a través de las conexiones directas. Eso no es del todo correcto, a menos que el 74x138 sea reemplazado con un decodificador con salidas de colector abierto. Explique por qué.

10.7 Describa la función lógica de siete variables que es efectuada por la ROM de 128×1 en la figura 10-7. Comenzando con el patrón de ROM, una manera de describir la función lógica es escribir la correspondiente tabla de verdad y suma canónica. Sin embargo, puesto que la suma canónica tiene 64 términos de producto de 7 variables, usted puede desear buscar una simple pero precisa descripción en palabras de la función.

10.8 Para la función lógica de dos salidas de la figura 4-39(b), compare el número de diodos o transistores necesarios en las realizaciones de AND-OR, AND, o arreglo de almacenamiento para PLA, PAL y ROM.

10.9 Muestre cómo duplicar el número de diferentes cantidades de atenuación que pueden seleccionarse en el atenuador digital de la figura 10-17 sin incrementar el tamaño de la ROM.

10.10 Escriba las funciones en lenguaje C `UlawToLinear` y `LinearToULaw` que se requieren en la tabla 10-6. En `LinearToULaw`, usted debería seleccionar el byte de ley μ cuyo valor teórico se encuentre más cercano a la entrada lineal. (*Sugerencia:* El enfoque más eficiente construye, en la inicialización del programa, una tabla de 256 entradas que se utiliza por ambas funciones.) Ambas funciones deberían informar de las entradas fuera de intervalo.

10.11 Modifique el programa en C de la tabla 10-6 para efectuar recorte en los casos donde el diseñador haya especificado una cantidad de atenuación particular para ser una ganancia, y al multiplicar un valor de entrada por el factor de atenuación produce un resultado fuera de intervalo.

10.12 Escriba un programa en C para generar el contenido de una ROM de 256×8 que haga la conversión de 8 bits binarios a 8 bits en código Gray. (*Sugerencia:* su programa debería incorporar el segundo método de construcción de código Gray de la sección 2.11.)

10.13 Escriba un programa en C para generar el contenido de una ROM de 256×8 que haga la conversión de 8 bits en código Gray a 8 bits binarios. (*Sugerencia:* se pueden aplicar los resultados del ejercicio 10.12. No importa si su programa es lento.)

10.14 Cierto sistema de comunicaciones ha sido diseñado para transmitir caracteres ASCII en forma serie en un medio que requiere de un nivel de señal promedio de cero, de manera que se utiliza un código de 5 a 10 para codificar los datos. Cada carácter de entrada ASCII de 7 bits es transmitido como una palabra de 10 bits con cinco ceros y cinco unos. Escriba un programa en C para generar el contenido de una ROM de 128×10 que convierta los caracteres ASCII en palabras codificadas.

10.15 El extremo receptor del sistema en el ejercicio 10.14 debe convertir cada palabra codificada de 10 bits de vuelta en un carácter ASCII de 7 bits. Escriba un programa en C para generar el contenido de una ROM de $1K \times 8$ que convierta palabras codificadas en caracteres ASCII. El bit de salida extra sería una marca o bandera de "error" que indicaría cuándo una palabra no perteneciente al código haya sido recibida.

10.16 ¿Cuántos bits de ROM se requerirían para construir un sumador/restador de 16 bits con control de modo, entrada de acarreo, salida de acarreo y salida de desbordamiento de complemento a dos? Sea más específico y no diga que son "miles y miles de millones"; explique su respuesta.

10.17 Repita el ejercicio 10.16 suponiendo que usted puede utilizar *dos* ROM, de modo que el retardo para una suma o resta de 16 bits sea dos veces el retardo a través de una ROM. Suponga que las dos ROM deben ser idénticas tanto en dimensiones como en programación. Intente minimizar el número total de bits de ROM requeridos y trace el circuito resultante sumador/restador. ¿Puede el número de bits de ROM ser reducido adicionalmente si se permite que las dos ROM sean diferentes?

10.18 Muestre la manera en que utilizando partes adicionales SSI/MSI, un 2764 puede ser usado como una ROM de $64K \times 1$. ¿Cuál es el tiempo de acceso de la ROM de $64K \times 1$?

10.19 Muestre la manera en que utilizando partes adicionales SSI/MSI, un 2764 puede ser usado como una ROM de $2K \times 32$. Se puede suponer la existencia de una señal de reloj de carrera libre cuyo periodo es ligeramente mayor que el tiempo de acceso del 2764. ¿Cuál es el tiempo de acceso de la ROM de $2K \times 32$?

10.20 Muestre la forma de construir el circuito sumador de ley μ de la figura 10-18 con una ROM de $32K \times 8$ y dos compuertas XOR. También, escriba un programa en C para generar el contenido de la ROM.

10.21 Determine el tamaño de la ROM necesaria para construir el codificador de punto fijo a punto flotante de la figura 6-3. Dibuje un diagrama lógico utilizando una ROM comercialmente disponible.

10.22 Escriba un programa en C para generar el contenido de la ROM para el ejercicio 10.21. A diferencia de la solución original MSI, su programa en C debería efectuar redondeo; es decir, para cada número de punto fijo se debería generar el número de punto flotante más cercano posible.

10.23 Dibuje un diagrama lógico completo para un circuito basado en ROM que efectúe la multiplicación combinacional de un par de enteros de 8 bits sin signo o de complemento a dos. La operación con signo o sin signo debería seleccionarse mediante una entrada simple, SIGNED. Se puede utilizar cualquiera de las ROM comerciales en la figura 10-11, además de compuertas discretas.

10.24 Escriba y pruebe un programa en C que genere el contenido de la(s) ROM(s) en el ejercicio 10.23.

10.25 Escriba un programa en C que genere una ROM de $256K \times 4$ que calcule el siguiente movimiento en un juego de "Tres en raya" (gato), empleando las codificaciones de entrada y salida de la sección 6.2.7. Su programa debe ser lo suficientemente listo para seleccionar un movimiento ganador siempre que sea posible.

10.26 Repita el ejercicio 10.25 utilizando una ROM de 32K × 4. Para conseguir esto, el estado del tablero debe ser codificado en sólo 15 bits. Explique su algoritmo de codificación, y escriba funciones en C para traducir un número de celda en cualquier dirección entre su codificación y la codificación de la sección 6.2.7.

10.27 Para cada uno de los parámetros de temporización definidos en la sección 10.3.3, determine si su valor para la SRAM de 128K × 8 que usted diseñó en el problema propuesto 10.5 es el mismo que el del HM628512. Si es diferente, indique el nuevo valor. Utilice los valores en el peor de los casos en la columna del 74FCT de la tabla 5-3 para determinar los retardos para la parte MSI.

10.28 Utilizando una SRAM HM6264 de 8K × 8, un puñado de partes MSI y un PLD como bloques de construcción, diseñe una SSRAM de escritura diferida de 8K × 8 con salidas de flujo continuo.

10.29 Utilizando una SRAM HM6264 de 8K × 8, un puñado de partes MSI y un PLD como bloques de construcción, diseñe una SSRAM ZBT de 8K × 8 con salidas canalizadas.

10.30 Defina los parámetros de temporización relevantes para la SRAM sicrónica como se definó en el ejerciccio 10.28.

10.31 Calcule los valores de los parámetros de temporización que se definieron en el ejercicio 10.30 para su solución al ejercicio 10.28.

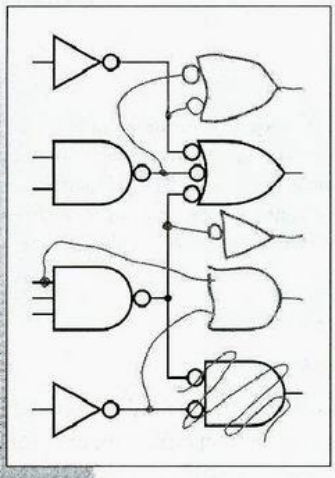
10.32 Por el estilo de la figura 10-27, dibuje el diagrama de temporización para una SSRAM de escritura diferida con salidas de flujo continuo para una serie de lecturas y escrituras intercaladas con el patrón R-R-W-W-R-W-R-W. Ejecute los ciclos individuales tan juntos como sea posible, pero asegúrese de tener en cuenta los conflictos de recurso que eviten los ciclos back-to-back. ¿Cuál es la utilización promedio del arreglo SRAM si la SSRAM se presenta con un flujo continuo de solicitudes R-W-R-W-R-W?

10.33 Repita el ejercicio anterior para una SSRAM de escritura diferida con salidas canalizadas.

10.34 Aplicando uno de los teoremas de la sección 4.1, demuestre que el CLB de la serie XC4000 de Xilinx puede realizar cualquier función lógica de cinco variables.

10.35 Muestre cómo utilizar los generadores de función en un CLB serie XC4000 para realizar una función de paridad par de 9 bits.

10.36 Muestre cómo utilizar los generadores de función en un CLB serie XC4000 para realizar un verificador de igualdad para dos operandos de 4 bits. Usted debería ser capaz de poner en cascada su diseño para verificar operandos de $4n$ bits con sólo n CLB.



capítulo 11

Temas adicionales del mundo real

Este capítulo incluye una colección de temas del "mundo real" que seguramente le interesarán a usted en algún momento si es (o quiere llegar a ser) un diseñador digital. Considere que analizaremos superficialmente cada una de estas áreas temáticas, así que asegúrese de consultar las referencias que pertenecen a las áreas que le interesan.

11.1 Herramientas de diseño asistido por computadora

"Si no era difícil, no podía llamarse hardware"

Muchos diseñadores digitales que tienen veinte años de experiencia consideran que esta afirmación es algo incuestionable. Conforme pasa el tiempo el diseño digital utiliza cada vez más las herramientas de *software*, y como consecuencia se está haciendo más fácil.

Los términos *diseño asistido por computadora (CAD, computer-aided design) e ingeniería asistida por computadora (CAE, computer-aided engineering)* se utilizan para indicar hacer las herramientas de software que ayudan a desarrollar circuitos, sistemas y muchas otras cosas. "CAD" es el término más general, y se aplica a las herramientas que se utilizan en la electrónica y en otras disciplinas, incluye las herramientas de diseño arquitectónico y mecánico. Por ejemplo, en el ámbito de la electrónica, "CAD" se refiere a herramientas de *diseño físico*, tales como programas de composición de CI y tarjetas de circuito impreso (PCB). "CAE" se usa con mayor frecuencia para indicar las herramientas de *diseño conceptual*, tales como editores de diagramas esquemáticos, simuladores de circuitos y compiladores PLD. Sin embargo, mucha gente que se desenvuelve en el campo de la electrónica (incluyendo al autor) tiene la tendencia a utilizar los dos términos en forma intercambiable. En esta sección, discutiremos algunas herramientas CAD/CAE que utilizan los diseñadores digitales.

¿ES AHORA EL HARDWARE "FÁCILWARE"?

Desde que más y más diseño y depuración de hardware se está llevando a cabo aplicando las herramientas de software, ¿se está en realidad haciendo más fácil? No necesariamente.

Por lo que ha experimentado el autor, el incremento en el uso de CAD significa que en lugar de gastar tiempo luchando con los cautines y las puntas de prueba, muchos diseñadores pueden ahora pasar más tiempo luchando con errores de programación en un ambiente de errores de software.

11.1.1 Lenguajes de descripción de hardware

En las décadas pasadas, la mayor parte del diseño lógico se efectuaba de manera gráfica, mediante el uso de diagramas de bloques y esquemáticos. Sin embargo, el surgimiento de los lenguajes de descripción de hardware ligado a los dispositivos de lógica programable y la tecnología ASIC a gran escala en la década de 1990 modificó radicalmente la manera en que se realizan los diseños digitales más grandes.

En el diseño de software tradicional los lenguajes de programación de alto nivel como C, C++ y Java han aumentado el nivel de abstracción, de modo que los programadores pueden diseñar sistemas más grandes y más complejos con algún sacrificio en el desempeño, en comparación con los programas en lenguaje ensamblador hechos a mano. La situación para el diseño de hardware es semejante. El circuito que produce una herramienta de síntesis de VHDL o Verilog puede no ser tan pequeño o rápido como uno que fue diseñado y fabricado a mano por un diseñador con experiencia, pero en las manos correctas estas herramientas pueden respaldar diseños de sistemas mucho más grandes. Claro que esto es un requerimiento si vamos a aprovechar los millones de compuertas que ofrecen las tecnologías más avanzadas CPLD, FPGA y ASIC.

El uso de los HDL se ha incrementado gracias a la disponibilidad de las herramientas de síntesis, pero los lenguajes no sintetizables ya existían desde hace algún tiempo. Los más prominentes son los *lenguajes de transferencia de registro,* que se han utilizado durante décadas para describir el funcionamiento de los sistemas sincrónicos. Un lenguaje de este tipo combina la notación de flujo de control de un lenguaje de descripción de máquina de estado con un medio para definición y la operación de registros de bits múltiples. Los lenguajes de transferencia de registro han sido especialmente útiles en el diseño de computadoras, donde las instrucciones individuales de lenguaje de máquina se definen como una secuencia de los pasos más primitivos que involucran los registros de carga, almacenamiento, combinación y prueba.

lenguajes de transferencia de registro

11.1.2 Captura de diagramas esquemáticos

Con excepción de las tareas, su primer paso en el diseño de un circuito es convencer a alguien que su enfoque propuesto es el adecuado. Eso significa preparar diagramas de bloques y diapositivas de presentación, además de discutir sus ideas con administradores y colegas en un *resumen preliminar del diseño*. Una vez que su proyecto ha sido aprobado, se puede comenzar con seguridad con el "asunto divertido", es decir, dibujar el diagrama esquemático.

repaso de diseño

Los diagramas se dibujaban a mano, pero la mayoría se preparan ahora con la ayuda de *editores de diagramas esquemáticos*, que son programas CAD que se ejecutan en estaciones de trabajo de ingeniería. El proceso de crear un diagrama en una computadora se conoce a menudo como *captura del diagrama esquemático.* (¿De dónde salió ese término?) Este

editor de diagramas esquemáticos

captura de diagramas esquemáticos

¿QUÉ ES UNA ESTACIÓN DE TRABAJO? Una "estación de trabajo de ingeniería" en la actualidad puede ser algo entre una PC de $500 USD hasta un procesador en paralelo de $50,000 USD. La estación de trabajo "canónica" en términos de costos (lo que una compañía típica gastará por ingeniero) y rendimiento (entre las mejores) es una estación de trabajo Sun de $5,000 USD, cualquiera que sea el modelo que usted obtenga por ese precio este año.

término se utiliza porque el editor de diagramas esquemáticos captura una mayor cantidad de información, que no se limita diagrama. La mayor parte de la información que se indica en el diagrama esquemático tiene un "tipo" asociado con ella, esto permite recuperar en forma automática la información que se selecciona en cualquier etapa durante el proceso de diseño (por ejemplo los valores de las componentes, su designación, localización, etcétera).

A fin de simplificar tanto la captura como la recuperación de la información que contiene un diagrama esquemático, los editores de diagramas esquemáticos proporcionan *campos* para cada tipo de información, ya sea que estén ocultos o identificados en forma automática cerca del elemento asociado. A continuación describiremos algunos campos típicos y sus aplicaciones:

campos

- *Tipo de componente.* Al especificar un tipo de componente (por ejemplo, resistor, capacitor, 74FCT374, GAL16V8), el diseñador puede invocar un símbolo predeterminado de una *librería de componentes*. Algunos componentes tienen símbolos alternos que puede seleccionar el usuario (por ejemplo, equivalentes de DeMorgan para las compuertas). El tipo componente se utiliza para propósitos de documentación del proceso de fabricación y de simulación.

tipo componente

librería de componente

- *Valor del componente.* La mayor parte de los componentes analógicos tienen un valor que se debe especificar en este campo (por ejemplo, 2.7 kΩ). Puede aparecer información distintiva adicional en un campo de "clasificación" o "tolerancia" (por ejemplo, 1/4 W, 1%). Los componentes digitales pueden proporcionar un valor nominal de velocidad aquí (por ejemplo, -10 para un PLD de 10 ns).

valor de componente

- *Número de parte aprobada.* Al utilizar toda la información distintiva de los campos anteriores, el software CAD puede seleccionar en forma automática un número de parte que está incluido en la "lista de partes aprobadas" de la compañía (por ejemplo, 126-10117-0272 para un resistor de película metálica de 2.7 kΩ, 1/4 W, 1%), o puede desplegar un aviso si no hay disponible ninguna parte de este tipo. (En las grandes compañías no se puede utilizar cualquier parte similar del catálogo; todas las partes deben tener la aprobación necesaria mediante un proceso de *calificación de las partes*.)

número de parte aprobada

calificación de las partes

- *Indicador de referencia.* Esta etiqueta alfanumérica distingue entre múltiples ejemplos de piezas semejantes y se utiliza casi en todos lados.

indicador de referencia

- *Ubicación del componente.* Al aplicar un sistema de coordenadas, este campo puede indicar la posición física del componente en un ensamble de tarjetas de circuito impreso, simplificando la tarea de encontrarlo durante la etapa de depuración final. Naturalmente, la ubicación final del componente generalmente no se conoce durante el diseño inicial del circuito. Sin embargo, el software CAE que tiene la capacidad de *anotación de respaldo* puede insertar esta información en el diagrama esquemático cuando se ha definido completamente la localización de los componentes en la tarjeta de circuito impreso.

ubicación de componente

anotación de respaldo

número de terminal

- *Números que identifican las terminales de los componentes.* Éstos especifican la asignación de las terminales de señal en cada componente, y por lo regular se especifican previamente en el símbolo que se toma de la biblioteca de componentes. Sin embargo, para las partes de integración a pequeña escala (SSI) y las que tienen varias secciones idénticas en un solo encapsulado, los números de terminales (incluso los indicadores de referencia) se especifican hasta que se ha terminado la ubicación de los componentes en la tarjeta de circuito impreso. Éste es otro caso donde resulta útil la anotación de respaldo.

tipo alambrado

- *Tipo de alambrado.* Los únicos tipos de alambrado son "ordinario" y "bus". Un bus es solamente un conjunto de alambres normales que se agrupan para facilitar la comprensión del diagrama. Sin embargo, en ciertos sistemas se pueden indiciar otros tipos de alambrado, como son las pistas de PCB extra anchas que transportan señales analógicas de alta corriente; esta información se transmite al programa que diseña la ubicación de los componentes en la tarjeta de circuito impreso.

nombre de señal

- *Nombre de señal.* El usuario puede (y debería) asociar un nombre a cada señal. El nombre es especialmente útil en las actividades de simulación y depuración, pero no afecta el diseño físico de la tarjeta de circuito impreso.

marca o bandera de conexión

- *Marcas de conexión.* Este elemento del diagrama indica la conexión de una línea de señal con otra que está en la misma página del diagrama o en otra diferente. El software puede asegurar que todas las conexiones de salida concuerden con las conexiones de entrada. En un esquema jerárquico, el software puede utilizar las marcas o banderas de conexión para generar nombres de señal en forma de un "símbolo lógico" que representa la página entera.

Una vez que usted ha "capturado" el diagrama esquemático de un circuito en un sistema CAD, hay muchas cosas que se pueden hacer con él, además de imprimirlo. Al menos se pueden generar dos documentos (a partir del diagrama esquemático) para propósitos de fabricación:

lista de partes

- *Lista de partes.* Ésta es una lista de componentes y sus indicadores de referencia.

red
lista de red

- *Lista de red.* Una *red* es un conjunto de terminales que se conectan al mismo nodo o señal eléctricos. Una *lista de red* especifica todas las conexiones que se necesitan en el diagrama, en general se utiliza una lista de nombres de señal que está ordenada alfabéticamente. Para cada nombre de señal, se indican las terminales del dispositivo (que están identificadas por el indicador de referencia y el número de terminal) a las que se conecta esa señal.

  De forma alternativa, la lista de red puede ordenarse con base en el indicador de referencia y el número de terminales, y puede indicar el nombre de la señal que se conecta en cada terminal; en ocasiones esto se conoce como una *lista de terminales.*

lista de terminales

La lista de partes y la lista de red son las entradas principales para el proceso de diseño y fabricación de la tarjeta de circuito impreso. El diseñador también puede utilizar la lista de partes para estimar el costo y confiabilidad del circuito, y el departamento de fabricación obviamente puede emplearla para adquirir las partes.

11.1.3 Especificaciones y diagramas de temporización

Los diagramas a bloques y esquemáticos no son los únicos diagrams que se utilizan en el diseño de un sistema digital. Los diagramas de temporización son una parte esencial de casi cualquier paquete de documentación.

Los sistemas más grandes están integrados por subsistemas más pequeños que se comunican a través de interfaces bien definidas. Normalmente la definición de una interfaz contiene además de los nombres de las señales y sus funciones, las especificaciones del comportamiento de temporización deseado. El punto de partida es típicamente una especificación de frecuencias máximas de reloj y en ocasiones se incluyen las frecuencias mínimas de reloj. Los requerimientos para los tiempos de establecimiento y retención de las señales de entrada de un subsistema se establecen con base en un flanco específico de reloj.

Los programas tales como TimingDesigner (`www.chronology.com`) pueden automatizar la tediosa tarea de crear complicados diagramas y especificaciones de temporización. En esta documentación los cambios en un parámetro de temporización pueden propagarse a muchos otros.

11.1.4 Análisis y simulación del circuito

La biblioteca de componentes de un sofisticado sistema CAD contiene más que un símbolo para cada componente. Para los CI, la biblioteca puede contener un *modelo de componente* que describe el funcionamiento eléctrico y lógico del dispositivo. De hecho, en un principio, los lenguajes de descripción de hardware como el VHDL y el Verilog se crearon estrictamente para modelar el comportamiento lógico de los componentes y su interconexión en los sistemas. Al aplicar estos modelos, los errores lógicos y de temporización se pueden encontrar durante la etapa de simulación.

modelo de componente

Como un mínimo, el modelo para un CI indica si cada terminal es una entrada o una salida. Con sólo esta información, un *verificador de regla de diseño* puede detectar varios de los errores más comunes "errores tontos" de un diseño, por ejemplo salidas en corto y entradas flotantes. Si se añaden las características de carga de entrada y excitación de salida para cada terminal, el programa puede determinar si se excede la cargabilidad de salida.

verificador de regla de diseño

El siguiente paso es la *verificación de temporización*. Incluso si no se tiene un modelo detallado del comportamiento lógico del CI, la librería de los componentes puede proporcionar un valor de retardo en el peor de los casos para cada trayectoria de entrada a salida, y los tiempos de establecimiento-retención de los dispositivos temporizados. Aplicando esta información, un *verificador de temporización* puede hallar las trayectorias de retardo en el peor de los casos para todo el circuito, en consecuencia el diseñador puede determinar si los márgenes de temporización son adecuados.

verificación de temporización

verificador de temporización

Por último, la biblioteca puede contener un modelo detallado del comportamiento lógico de cada componente, en cuyo caso puede utilizarse la *simulación* para predecir el comportamiento de todo el circuito ante cualquier secuencia de entradas. El diseñador proporciona una secuencia de entrada, y el programa *simulador* determina cómo responderá el circuito a esa secuencia. Normalmente la salida del simulador se exhibe gráficamente, mediante formas de onda de temporización que el diseñador podría ver en un osciloscopio o analizador lógico (como si se aplicaran las mismas señales si se aplican las mismas señales a un circuito real). En un ambiente así, se puede depurar un circuito completo sin tener que "construirlo" y ensamblalo en una tarjeta de circuito impreso que funcione realmente al primer intento.

simulación

simulador

Los simuladores utilizan modelos diferentes para diferentes tipos de componentes. Los simuladores analógicos como SPICE pueden utilizar modelos matemáticos con sólo un parámetro para un resistor, y pueden utilizar desde unos cuantos hasta docenas de parámetros para un transistor bipolar. Basado en estos modelos y la estructura del circuito, el simulador desarrolla y resuelve ecuaciones para determinar el comportamiento del circuito.

Los simuladores digitales normalmente utilizan uno de los dos modelos siguientes para cada elemento lógico:

modelo de comportamiento

- *Modelo de comportamiento (o modelo basado en el comportamiento).* En un modelo de esta clase, el software "entiende" la función básica del elemento lógico. Por ejemplo, se entiende que una compuerta AND producirá una salida 1 con retardo t_{pHL} después que todas sus entradas sean 1 y producirá una salida 0 con retardo t_{pHL} después que cualquier entrada sea 0.

modelo estructural

- *Modelo estructural.* Los elementos más grandes pueden ser modelados al conectar grupos de elementos más pequeños que tengan modelos de comportamiento.

Anteriormente vimos que el lenguaje VHDL respalda ambos tipos de modelos. Por ejemplo, la tabla 5-15 presenta un modelo de comportamiento para un decodificador 74x138 de 3 a 8. Al utilizar una de las cláusulas VHDL ("`after`" y otros mecanismos que no se indican en este libro), podríamos especificar el retardo de la entrada a la salida. Alternativamente, la tabla 5-14 proporciona el modelo estructural de un decodificador de 2 a 4 como un conjunto de compuertas que están conectadas de acuerdo al diagrama lógico interno de la figura 5-32. Suponiendo que ya existe la información de temporización para las compuertas individuales de la estructura (`inv` y `and3`), el modelo estructural más grande puede predecir la temporización del circuito con más precisión, puesto que puede seguir la trayectoria del circuito real que toma cada transición diferente de entrada.

modelo de software

Ambos modelos se conocen como *modelos de software* debido a que el funcionamiento del chip es simulado enteramente mediante un software. Algunos chips, incluyendo los microprocesadores y otras partes tipo LSI, requieren de enormes modelos de software. En la mayoría de los casos, los fabricantes del chip no proporcionan tales modelos (ya que los consideran información confidencial), además un usuario necesitaría meses o años para desarrollar los modelos. Para evitar este problema, las compañías innovadoras de CAD han desarrollado un tercer tipo de modelo:

modelo físico
modelo de hardware

- *Modelo físico*, que en ocasiones se denomina *modelo de hardware*. La copia real y funcional de un chip se conecta a la computadora que ejecuta la simulación. El simulador aplica entradas al chip real y observa sus salidas, las cuales se utilizan como entradas para otros modelos en la simulación.

La entrada de un diseñador digital a un simulador lógico es el diagrama esquemático y una secuencia de vectores de entrada que se aplicarán al circuito en tiempo simulado. La primera salida que observa el diseñador es "xxxxx", lo que significa que el simulador no puede determinar cuál será la salida. Esto sucede con bastante frecuencia porque el circuito incluye circuitos secuenciales (flip-flops y latchs) que no están inicializados. Para obtener una salida que tenga sentido en el simulador, el diseñador debe comprobar que todos los circuitos estén explícitamente restablecidos a un valor conocido, aun cuando no se necesite el reestablecimiento para obtener el funcionamiento adecuado del sistema real. Por ejemplo, los contadores de anillo de autocorrección de las figuras 8-62 y 8-64 no se pueden simular porque el simulador no puede determinar cuál es su estado inicial.

circuitos secuenciales sin inicializar

Una vez que se ha inicializado el circuito simulado, la capacidad del diseñador para probar su funcionamiento se ve limitada solamente por la velocidad del simulador. La simulación de circuitos complejos puede ser muy lenta, en comparación con la velocidad del objeto real. Dependiendo del nivel de detalle que se esté simulando, el tiempo de simulación puede ser de 10^3 hasta 10^8 veces mayor que el intervalo que se está simulando. Una simulación estrictamente funcional, en la que se supone que todos los componentes tienen un retardo igual a cero,

ineficiencia de simulación

es más rápida. Una simulación mucho más realista, que calcule y considere una amplia gama de retardos en el peor de los casos para todas las trayectorias de señal, es la más lenta.

11.1.5 Asignación de componentes en la tarjeta de circuito impreso

La ubicación de los componentes en una tarjeta de circuito impreso (PCB, printed-circuit-board) se puede realizar con la ayuda de un programa o una persona (un diseñador de PCBs) o mediante la combinación de ambos. En general los diseñadores de tarjetas de circuito impreso PCB son personas a las que les gusta resolver rompecabezas. Su trabajo consiste en acomodar todos los componentes del diagrama esquemático en una tarjeta que tiene unas dimensiones determinadas, y posteriormente conectar todas las conexiones como se requieren según el diagrama esquemático. Normalmente tienen que realizar lo anterior minimizando el tamaño de la tarjeta y la cantidad de caras de interconexión de la tarjeta de circuito impreso.

El diseño de una tarjeta de circuito impreso puede hacerse en forma automática mediante el software, pero los resultados no siempre son tan buenos. Tanto el diseñador de tarjetas como el diseñador del circuito normalmente deben guiar los pasos iniciales de la colocación de componentes, tomando en cuenta muchos factores prácticos, que incluyen los siguientes:

- *Limitantes mecánicas*. Ciertos conectores, indicadores y otros componentes deben instalarse en una ubicación especial sobre la tarjeta debido a ciertos requerimientos mecánicos predeterminados.

  Limitaciones mecánicas

- *Trayectorias críticas de señal.* Un programa de colocación automática puede situar los componentes en una manera que minimice la longitud promedio de las interconexiones, pero normalmente los diseñadores tienen un sentido (o intuición) que les permite saber qué áreas del circuito y cuáles señales específicas tienen los requerimientos más críticos. De hecho, es común que el diseñador del circuito proporcione un *plano de distribución* al diseñador de la tarjeta. Este plano sugiere la colocación de ciertos grupos de componentes que reflejan las opiniones del diseñador acerca de cuáles elementos necesitan estár más cerca de otros.

  trayectorias de señal crítica

  plano de distribución

- *Aspectos térmicos.* Algunas áreas del PCB pueden tener mejor enfriamiento que otras cuando se instala la tarjeta en un sistema, y ciertamente algunos componentes funcionarán más calientes que otros. Los diseñadores deben verificar que no se exceda el intervalo de temperatura de operación de los componentes.

  aspectos térmicos

  La temperatura en cualquier parte de una tarjeta que está bien diseñada, para un sistema que tiene el enfriamiento adecuado, no debe superar la temperatura ambiente por 10 °C. Sin embargo, si se instala un chip hambriento de energía (como un microprocesador) en una área donde no existe flujo de aire, la temperatura puede aumentar hasta 30 °C o más. Si la temperatura ambiente es de 40 °C, la temperatura real en el microprocesador sera 70 °C, que es el límite del intervalo (de la temperatura de operación) para la mayor parte de los chips.

- *Emisiones de radiofrecuencia.* El equipo electrónico emite energía de radiofrecuencia como un efecto secundario. Las agencias de gobierno de Estados Unidos de Norteamérica y otros países especifican el nivel máximo permisible de estas emisiones. (En Estados Unidos estos límites se explican con detalle en las normas de la *FCC parte 15.*)

  emisiones de radiofrecuencia

  FCC parte 15

  En general, los niveles de emisiones de un circuito dependen tanto del rendimiento y configuración del mismo, como de la ubicación de los componentes en la tarjeta. En la etapa de ubicación de los componentes en la tarjeta, se debe prestar especial

atención a la colocación de los componentes, las conexiones a tierra, aislamiento y desacoplamiento, con la finalidad de minimizar las emisiones no deseadas (¡e ilegales!) de radiofrecuencia.

errores tontos (u obvios)

- *Errores tontos.* Quizás el paso más importante en el diseño de una tarjeta, en especial cuando utilizamos componentes nuevos o componentes con los que estamos poco familiarizados, consiste en verificar que no cometamos errores tontos. Casi cualquier diseñador digital, incluso aquellos que tienen más experiencia, pueden comentarnos lo que sucedió en un circuito cuando al diseñar la ubicación de un componente en especial, utilizaron la imagen invertida de las terminales del componente, o cuando marcaron las perforaciones de montaje de un componente o conector en el lugar equivocado, cuando utilizaron un encapsulado diferente de un mismo CI, cuando utilizaron el extremo contrario de un conector, cuando dejaron sin conectar las capas internas de conexión de una tajeta de circuito impreso con varias capas, y demás comentarios por el estilo. Recordemos la frase que dice "las computadoras no cometen los errores, ¡son los humanos los que lo hacen!"

enrutamiento

Una vez que se ha efectuado la colocación de los componentes en la tarjeta, se deben conectar todas las señales; esto se denomina *enrutamiento*. Esta fase del diseño de tarjetas es mucho más sencilla y fácil de automatizar; de hecho, un "enrutador automático" puede enrutar una tarjeta de circuito impreso (PCB) de dimensiones moderadas durante la noche. De cualquier forma, puede ser que los ingenieros de diseño tengan que poner especial atención al enrutamiento de las pistas de una trayectoria crítica y otros aspectos como son la colocación de terminaciones de señal y puntos de prueba, los cuales se analizan más adelante en este capítulo.

11.2 Diseño para pruebas

Cuado usted adquiere una nueva pieza de equipo, espera que funcione adecuadamente "inmediatamente después que la saca de su empaque". Sin embargo, incluso con el moderno equipo automatizado, es imposible que un fabricante de equipo garantice que todas las unidades producidas serán perfectas. Algunas unidades no funcionarán adecuadamente porque contienen componentes individuales que son defectuosos, porque se ensamblaron en forma incorrecta, debido a daños que sufrieron durante las operaciones de almacenamiento y envío. La mayor parte de los fabricantes prefieren descubrir estos problemas en sus instalaciones, en vez de obtener mala reputación cuando sus clientes reciben un producto defectuoso. Éste es el propósito de las *pruebas*.

prueba
prueba de pasa/no pasa

La prueba más básica, una *prueba de pasa/no pasa* produce sólo un bit de información: ¿el sistema funciona al 100% o no? Si la respuesta es sí, el sistema se puede enviar al cliente. En caso contrario, la siguiente acción depende del tamaño del sistema. La respuesta apropiada para *un* reloj digital de pulsera sería tirarlo a la basura. Sin embargo, si *muchos* relojes presentan la falla, la mejor opción consiste en solucionar lo que parece ser un problema consistente en el proceso de ensamble o en los componentes individuales, en vez de tratar de recuperar las unidades individuales.

prueba de diagnóstico

Por otra parte, a usted no le gustaría tener que deshacerse de una computadora que costó $10,000 USD porque falla durante la inicialización. En vez de ello, puede ejecutar una *prueba de diagnóstico* más detallada para localizar el subsistema particular que está fallando. Dependiendo del costo, puede optar por reparar o reemplazar el subsistema defectuoso. Un subsistema digital típico es una tarjeta cuyo costo varía desde $50 hasta $1,000 USD. La

decisión de reparar o reemplazar es un compromiso económico entre el costo de la tarjeta completamente ensamblada y el tiempo que se necesita para localizar y reparar la falla.

diseño para pruebas (DFT)

Aquí es donde entra el *diseño para pruebas (DFT, design for testability)*. "DFT" es un término general que se aplica a los métodos de diseño que conducen a procesos de prueba más detallados y menos costosos. Se obtienen muchos beneficios cuando se diseña un sistema o subsistema de manera que las fallas sean fáciles de detectar y localizar:

- El resultado de una prueba de pasa/no pasa es más confiable. Si se producen menos sistemas con fallas ocultas, menos clientes quedarán disgustados, lo que producirá obviamente beneficios económicos así como sicológicos.
- Las pruebas de diagnóstico se ejecutan con mayor rapidez y producen resultados más precisos. Esto reduce los costos del retrabajo de un subsistema que falla en la prueba de pasa/no pasa, esto hace posible la fabricación de una mayor cantidad de sistemas a menor costo.
- El personal de ingeniería de pruebas necesita menos tiempo para desarrollar las pruebas del tipo pasa/no pasa y las pruebas de diagnóstico.
- Aunque los ahorros en el tiempo de ingeniería de pruebas se pueden compensar al realizar esfuerzos adicionales en la etapa de ingeniería-diseño para incluir el diseño para pruebas, cualquier incremento en el costo del desarrollo total del producto generalmente se puede compensar cuando disminuye el costo de los procesos de fabricación.

11.2.1 Pruebas

vector de prueba

Los circuitos digitales se verifican mediante la aplicación de *vectores de prueba* que consisten en combinaciones de entradas y salidas determinadas. Un circuito "pasa" si sus salidas coinciden con lo esperado. En el peor de los casos, un circuito combinacional de n entradas requiere de 2^n vectores de prueba. Sin embargo, si conocemos algo acerca de la construcción física del circuito y hacemos algunas suposiciones acerca del tipo de fallas que se pueden presentar, el número de vectores que se requieren para probar compeltamente el circuito puede ser reducirse en gran forma. La suposición más común es suponer que las fallas son *fallas simples y persistentes*, es decir, que se pueden modelar como una sola señal de entrada o salida que se estanca en un 0 lógico o 1 lógico. Bajo esta suposición, una compuerta NAND de 8 entradas, que de otro modo requiere de 256 vectores de prueba, puede ser completamente probada con solamente nueve: 11111111, 01111111, 10111111, ..., 11111110.

falla persistente simple

Bajo la suposición de falla simple, es fácil plantear vectores de prueba para elementos lógicos individuales. Sin embargo, el problema en la práctica es *aplicar* los vectores de prueba a elementos lógicos que se encuentran profundamente inmersos en un circuito, y *ver* los resultados. Por ejemplo, supongamos que un circuito tiene una docena de elementos lógicos combinacionales y secuenciales entre sus entradas primarias y las entradas de una compuerta NAND de 8 entradas que deseamos probar. No es tan obvio establecer cuál vector de entrada primaria, o secuencia de vectores de entrada primaria debe aplicarse para generar el vector de prueba 11111111, en las entradas de la compuerta NAND. Además, no es obvio qué más puede requerirse para propagar la salida de la compuerta NAND a una salida primaria del circuito.

programa de generación de pruebas

conjunto completo de pruebas

Los sofisticados *programas de generación de pruebas* tratan con esta complejidad e intentan crear un *conjunto completo de pruebas* para un circuito, es decir, una secuencia de patrones de prueba que prueben exhaustivamente cada elemento lógico en el circuito. Sin embargo, el cálculo requerido puede ser enorme, y con mucha frecuencia se descubre que no es posible generar un conjunto completo de prueba.

controlabilidad

observabilidad

puntos de prueba

Los métodos DFT intentan simplificar la generación de patrones de prueba mediante el mejoramiento de la "controlabilidad" y "observabilidad" de los elementos lógicos de un circuito. En un circuito con buena *controlabilidad*, es fácil producir cualquier valor deseado en las señales internas del circuito mediante la aplicación de una apropiada combinación de entradas de vectores de prueba en las entradas primarias. En forma similar, buena *observabilidad* significa que cualquier señal interna se puede propagar fácilmente a una salida primaria, para compararla con un valor deseado, mediante la aplicación de una combinación adecuada de la entrada primaria. El método más común para mejorar la controlabilidad y la observabilidad consiste en agregar *puntos de prueba,* entradas y salidas primarias adicionales que se utilizan durante el proceso de prueba.

11.2.2 Dispositivos de prueba y equipos de prueba en el circuito

dispositivo especial de prueba

terminales de contacto

equipo automático de prueba

prueba en el circuito

Si consideramos que los circuitos digitales se fabrican en una sola tarjeta de circuito impreso, las tendencias actuales en observabilidad indican el uso de cada una de las terminales de todos los CIs como puntos de prueba. Esto se consigue mediante el uso de un *dispositivo especial de prueba* (test fixture), cuyas *terminales de contacto* (pogo-pins) coinciden con los puntos de conexión de la tarjeta de circuito impreso que están en el lado de las pistas. Las terminales de contacto del dispositivo de prueba están formadas por un perno que tiene un resorte en su interior. El dispositivo de prueba tiene una terminal de contacto para cada patita o terminal de los circuitos integrados. La tarjeta que se va a probar se coloca en el dispositivo (que parece una "cama perforada" llena de pogo-pins) y a su vez, las terminales de contacto del dispositivo de prueba se conectan a un equipo automático de prueba que puede monitorear el estado de cada terminal como lo indica un programa de prueba (test software).

Avanzando un paso más, los equipos de *prueba en el circuito* también realizan lo "último" en controlabilidad. Este método no sólo monitorea las señales en el dispositivo de prueba, sino que también conecta cada terminal de contacto (del dispositivo de prueba) a un circuito de control de baja impedancia que forma parte del equipo automático de prueba. De esta forma, el equipo automático de prueba puede cancelar (o *sobreexcitar*) cualquier circuito de la tarjeta que normalmente controla cada señal, y generar directamente cualquier vector de prueba deseado en las señales internas de la tarjeta. Al sobreexcitar la salida opuesta de una compuerta

sobreexcitación

DANDO EN EL CLAVO (DE LA TERMINAL DE CONTACTO)

El incremento de los dispositivos que tienen encapsulados de alta densidad y montaje superficial ha provocado que las pruebas que utilizan dispositivos especiales (que contienen pogo-pins) sean considerablemente más difíciles de efectuar, si se comparan con los componentes cuyas terminales se instalan a través de los orificios de montaje de la tarjeta, p. ej. componentes con encapsulado tipo DIP.

Puesto que los componentes se pueden montar en ambas caras de la tarjeta de circuito impreso, se puede utilizar un dispositivo especial de prueba que se conoce como "armazón de almeja" ("clam shell") para conectar las terminales de prueba del dispositivo en ambos lados de la tarjeta. Además, las terminales de la mayoría de los dispositivos de montaje superficial son tan pequeñas y su separación es tan estrecha (25 milésimas de pulgada o menos) que la inserción de una terminal de contacto del dispositivo de prueba puede ser prácticamente imposible. En tales casos, el diseñador de la tarjeta tiene que proporcionar explícitamente isletas de prueba, que son secciones adicionales de las pistas de cobre que tienen las dimensiones suficientes para permitir el contacto de una terminal de prueba (por ejemplo, una isleta que tiene 50 milésimas de pulgada de diámetro). "Se debe proporcionar una isleta de prueba por separado para cada señal que se conecte a una la isleta de mayor tamaño o una perforación de montaje ("thru-hole") que tenga un diámetro de 62 milésimas de pulgada, en alguna parte de la tarjeta."

se genera un flujo de corriente excesivo tanto en el equipo de prueba como en la compuerta opuesta, pero el equipo de prueba la soporta durante breves instantes (milisegundos).

Para verificar una compuerta NAND de 8 entradas, un equipo de prueba en el circuito solamente necesita proporcionar los nueve vectores de prueba antes mencionados, y puede ignorar cualquier valor que el resto del circuito trata de controlar en las ocho terminales de entrada. La salida de la compuerta NAND puede monitorearse directamente en la terminal de salida. Con la prueba en el circuito, se puede probar cada elemento lógico en forma independiente.

Aunque la prueba en el circuito mejora en gran medida la controlabilidad y la observabilidad de un circuito que utiliza tarjetas de circuito impreso, los diseñadores lógicos todavía deben seguir varios lineamientos de diseño para pruebas, para lograr que el enfoque sea efectivo. A continuación se mencionan algunos lineamientos.

- *Inicialización.* Debe ser posible inicializar todos los elementos del circuito secuencial a un estado conocido.

Puesto que las terminales de entrada, preestablecimiento y borrado de los registros y flip-flops están disponibles para el equipo de prueba en el circuito, se podría pensar que éste no es un problema. Sin embargo, la figura 11-1(a) muestra el caso clásico de un circuito (un contador de código Gray) que no se puede inicializar, puesto que los flip-flops pasan a un estado impredecible cuando PR y CLR se afirman en forma simultánea. La manera correcta de manejar las entradas de preestablecimiento y borrado se muestra en (b).

- *Generación de señales de reloj.* El equipo de prueba debe ser capaz de proporcionar su propia señal de reloj *sin* sobreexcitar las señales del reloj de la tarjeta.

En general los equipos de prueba deben cancelar un reloj de la tarjeta (que se va a probar), por varias razones: la velocidad a la que pueden aplicar los vectores de prueba es limitada; por lo tanto deben esperar hasta que se nivelen las señales sobreexcitadas; y en ocasiones

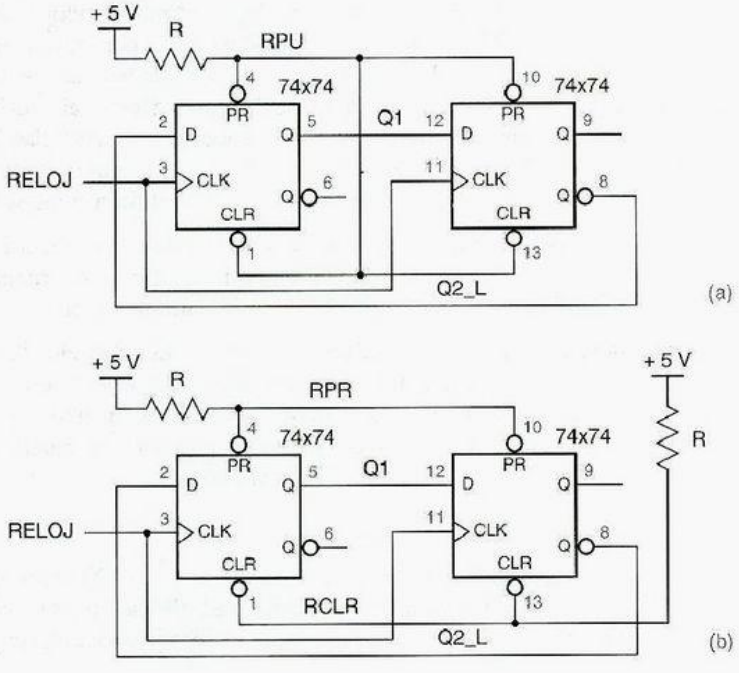


Figura 11-1
Flip-flops con resistores de arranque para entradas sin utilizar: (a) no factibles de probarse; (b) factibles de probarse.

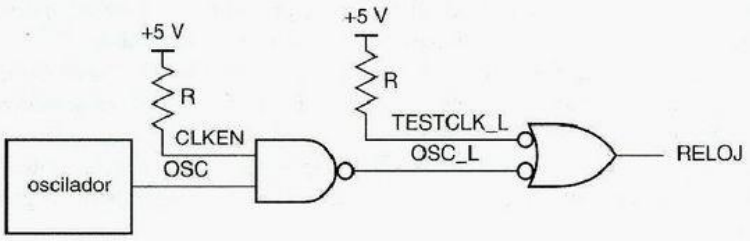


Figura 11-2
Circuito controlador de reloj que permite que un equipo de prueba cancele limpiamente el reloj del sistema.

deben detener el reloj. Sin embargo, sobrecontrolar (o sobreexcitar) el reloj está prohibido. Una señal sobreexcitada puede "Oscilar" y efectuar varias transiciones entre los niveles BAJO y ALTO antes de establecerse finalmente en el nivel que desea el equipo de prueba. En una señal de reloj, tales transiciones pueden crear cambios indeseables de estado.

La figura 11-2 muestra una opción más recomendable para un circuito controlador de reloj. Para inyectar su propio reloj, el equipo de prueba hace CLKEN LOW e inserta su reloj en TESTCLK_L. Como el equipo de prueba no sobreexcita ninguna salida de la compuerta, la señal CLOCK resultante es limpia. En general, el equipo de prueba no debe sobreexcitar a ninguna señal sin interferencias que se utiliza como entrada de reloj u otra entrada asincrónica, en teoría esta señal se tiene que tratar como se indica en la figura 11-2. Ésta es otra razón por la que es más aconsejable el diseño sincrónico con sólo un reloj.

- *Entradas aterrizadas.* En general, la tierra no debe utilizarse como una fuente para un 0 lógico y constante.

El equipo de prueba en el circuito puede sobreexcitar la *mayor parte* de las señales, pero no puede sobreexcitar la tierra. Por lo tanto, las señales que requieren una entrada constante en 0 —durante la operación normal del circuito— deben conectarse a tierra a través de un resistor. Esto permite que el equipo de prueba las establezca a 1 cuando sea necesario durante la prueba. Por ejemplo, considere lo que pasaría si utilizamos un PLD GAL16V8 para generar un conjunto personalizado de formas de onda de reloj a partir de un reloj maestro, como lo hicimos en el programa ABEL en la tabla 8-26. Si no fuera por consideraciones de prueba, podríamos conectar la terminal 11 del 16V8 —terminal de habilitación de salida global de nivel activo bajo— directamente a tierra. Sin embargo, esta terminal se debe conectar a tierra a través de un resistor, para que el equipo de prueba pueda flotar las salidas del PLD y controlar las señales de reloj P1_L–P6_L por sí mismo. Aunque el equipo de prueba podría teóricamente sobreexcitar estas señales, no debería hacerlo si se utilizan como señales de reloj.

- *Controladores de bus.* En general, se debe permitir la deshabilitación de los controladores para buses amplios, de modo que el equipo de prueba pueda controlar el bus sin tener que sobreexcitar todas las señales en el bus.

Es decir, se debe contar con la deshabilitación de la salida de todos los controladores de tres estados en el bus, de manera que el equipo de prueba controle un bus "flotante". Esto reduce la tensión eléctrica tanto en el equipo de prueba como en los controladores de bits múltiples (por decir un 74x244) que de otro modo pueden sobrecalentarse y sufrir daños si se sobreexcitan todas sus salidas en forma simultánea.

11.2.3 Métodos de exploración

La prueba en el circuito funciona bien, hasta cierto punto. No hace muy buen trabajo para ASIC y chips VLSI personalizados, porque sencillamente las señales internas no se encuentran accesibles. Aun en los circuitos que se contruyen a nivel tarjeta PCB, las tec-

nologías de encapsulado de alta densidad, como el montaje superficial, incrementan en gran medida la dificultad de proporcionar un punto de prueba para cada señal en una tarjeta de circuito impreso. Como resultado, un creciente número de diseños utilizan "métodos de exploración" para proporcionar controlabilidad y observabilidad.

método de exploración
método de trayectoria de exploración

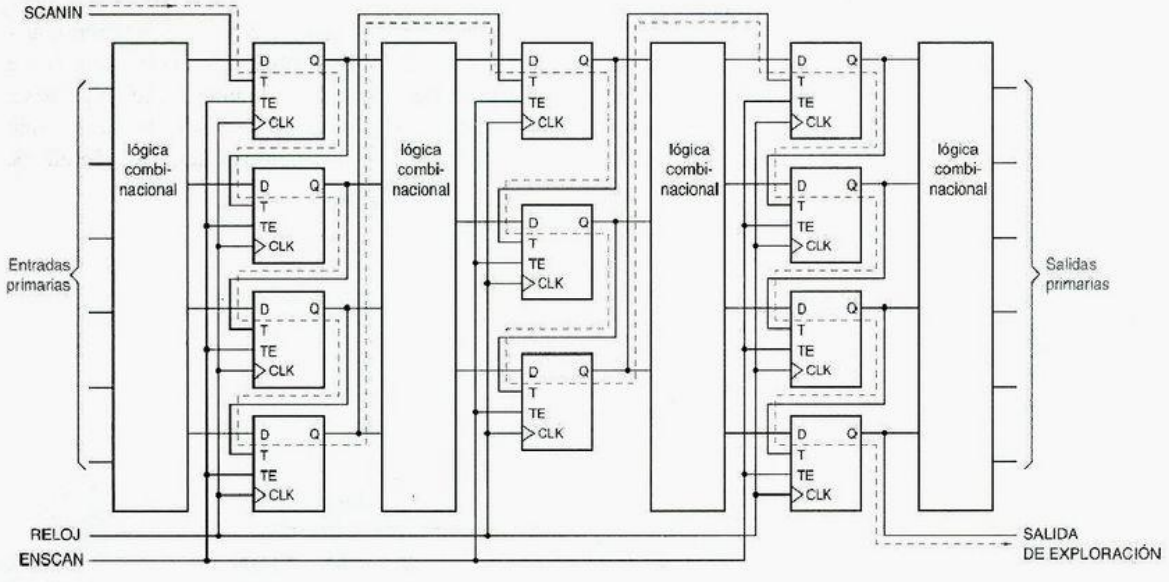
El *método de exploración* intenta controlar y observar las señales internas de un circuito, para tal fin utiliza solamente un pequeño número de puntos de prueba. El método de *trayectorias de exploración* considera que cualquier circuito digital es una colección de flip-flops u otros elementos de almacenamiento que están interconectados mediante la lógica combinacional. El interés de este método abarca el control y la observación del estado de los elementos de almacenamiento. Lo anterior se realiza al proporcionar dos modos de operación: un modo normal, y un *modo de exploración* en el que todos los elementos de almacenamiento se reorganizan en un registro gigante de corrimiento. En el modo de exploración, el estado de los n elementos de almacenamiento del circuito se puede obtener mediante n corrimientos (*observabilidad*), y se puede cargar un estado nuevo al mismo tiempo (*controlabilidad*).

modo de exploración

La figura 11-3 muestra el diseño de un circuito que utiliza un método de trayectoria de exploración. Cada elemento de almacenamiento de este circuito es un flip-flop de exploración (sección 7.2.7) que puede cargarse desde una de ambas fuentes. La entrada de habilitación de prueba (TE, test enable) selecciona la fuente: datos normales (D) o datos de prueba (T). Las entradas T están encadenadas para crear la trayectoria de exploración que se muestra con líneas discontinuas. Al activar ENSCAN para 11 tics de reloj, un equipo de prueba puede leer el estado actual de los flip-flops y cargar un nuevo estado. Se deja al ingeniero de pruebas el trabajo de obtener los conjuntos de prueba para los bloques lógicos combinacionales individuales, los cuales pueden ser totalmente controlados y observados utilizando la trayectoria de exploración y las entradas y salidas primarias.

El diseño de la trayectoria de exploración se utiliza con frecuencia en el diseño personalizado de VLSI y ASIC, debido a la imposibilidad de proporcionar un gran número de puntos de prueba convencionales. Sin embargo, los flip-flops de dos puertos empleados

Figura 11-3 Circuito conteniendo una trayectoria de exploración, mostrada con línea punteada.



en el diseño de trayectoria de exploración incrementan el área del chip. Por ejemplo, en la serie LCA500K de arreglos de compuertas CMOS de LSI Logic Corp., una macrocelda flip-flop D FD1QP utiliza 7 "celdas de compuerta", mientras que una macrocelda flip-flop de exploración D FD1SQP hace uso de 9 celdas de compuerta, casi un 30% de incremento en el área de silicio. Sin embargo, el incremento total en el área de chip es mucho menor, puesto que los flip-flops son solamente una fracción del chip, y las estructuras de memoria "regular" más grandes (por ejemplo, RAM) pueden probarse por otros medios. En cualquier caso, el mejoramiento en la capacidad de prueba puede realmente *reducir* el costo del chip encapsulado cuando se considera el costo de las pruebas. Para diseños ASIC grandes que tienen complejas estructuras de control, el diseño de trayectoria de exploración debería ser considerado como un requerimiento.

11.3 Estimación de la confiabilidad del sistema digital

confiabilidad

Cualitativamente, la *confiabilidad* de un sistema digital es la probabilidad de que funcione correctamente cuando usted lo necesita. Al personal de mercadotecnia y de ventas le gusta mencionar que los sistemas que distribuyen tienen una "alta confiabilidad", lo cual significa que tienen "bastante probabilidad de mantenerse en funcionamiento". Sin embargo, los clientes con sentido práctico hacen preguntas que requieren respuestas más concretas, tales como "Si adquiero 100 de estos sistemas, ¿cuántos fallarán en el transcurso de un año?" Para proporcionar estas respuestas, los ingenieros en diseño digital tienen la responsabilidad de calcular la confiabilidad de los sistemas que diseñan, y en cualquier caso deberían estar conscientes de los factores que afectan esta confiabilidad.

Cuantitativamente, la confiabilidad se expresa como una función matemática del tiempo:

$R(t)$ = Probabilidad de que el sistema aún funcione correctamente al tiempo t.

La confiabilidad es un número real entre 0 y 1; es decir, en cualquier instante $0 \leq R(t) \leq 1$. Suponemos que $R(t)$ es una función linealmente creciente; es decir, que las fallas son permanentes y no consideramos los efectos de la reparación. La figura 11-4 es una típica función de confiabilidad.

La definición precedente de confiabilidad supone que se conoce la definición matemática de probabilidad. Si no es así, la confiabilidad y la probabilidad correspondiente son más fáciles de definir en términos de un experimento. Supongamos que fuéramos a construir y hacer funcionar N copias idénticas del sistema en cuestión. Hagamos que $W_N(t)$ denote el número de ellas que todavía estarían funcionando al tiempo t. Entonces,

$$R(t) = \lim_{n \to \infty} W_N(t)/N$$

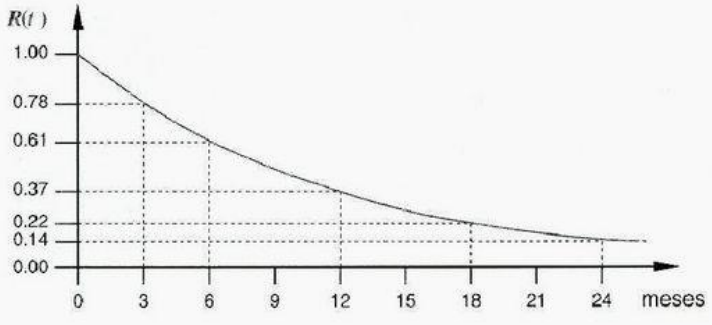


Figura 11-4
Función de confiabilidad típica para un sistema.

Es decir, si construimos *muchos* sistemas, $R(t)$ es la fracción de los mismos que todavía se encontrarán funcionando al tiempo t. Cuando hablamos acerca de la confiabilidad de un solo sistema, simplemente aplicamos nuestra experiencia a una población elevada para estimar nuestras oportunidades con una sola unidad.

Sería muy costoso calcular $R(t)$ si la única manera de hacerlo fuera efectuar un experimento (construir y monitorear N copias del sistema. Peor aún, para cualquier t, no sabríamos el valor de $R(t)$ hasta que haya transcurrido el tiempo t en tiempo real. De este modo, para responder la pregunta del cliente que se planteó con anterioridad, tendríamos que construir un manojo de sistemas y esperar un año; para entonces, nuestro cliente potencial habría adquirido otra cosa.

En cambio, podemos estimar la confiabilidad de un sistema combinando la información de confiabilidad de sus componentes individuales, aplicando un modelo matemático simple. La confiabilidad de los componentes maduros (por ejemplo, chips 74FCT CMOS) puede conocerse y publicarse basada en evidencia experimental real, mientras que la confiabilidad de los nuevos componentes (por ejemplo, un microprocesador Sextium) puede estimarse o extrapolarse a partir de la experiencia con alguno semejante. En cualquier caso, la confiabilidad de un componente se describe típicamente mediante un solo número, el "índice de fallas" que se describe a continuación.

11.3.1 Índice de fallas

índice de fallas

El *índice de fallas* es el número de fallas que se presentan por unidad de tiempo en un componente o sistema. En fórmulas matemáticas, el índice de fallas se denota habitualmente por la letra griega λ. Puesto que las fallas se presentan con poca frecuencia en equipo electrónico, el índice de fallas es medido o estimado utilizando muchas copias idénticas de un componente o sistema. Por ejemplo, si hacemos funcionar 10,000 chips microprocesadores durante 1,000 horas, y fallan ocho de ellos, diríamos que el índice de fallas es

λ

$$\lambda = \frac{8 \text{ fallas}}{10^4 \text{ chips} \cdot 10^3 \text{ horas}} = (8 \cdot 10^{-7} \text{ fallas/hora})/\text{chip}$$

Es decir, el índice de fallas de un chip simple es $8 \cdot 10^{-7}$ fallas/hora.

El proceso real de estimación para la confiabilidad de un lote de chips no es ni con mucho tan sencillo como acabamos de describir; para más información, consulte las Referencias. Sin embargo, podemos emplear los índices de fallas de componentes individuales, que se obtienen con cualquier medio, en un modelo matemático directo para predecir la confiabilidad total del sistema, como lo mostraremos más tarde en esta sección.

Puesto que los índices de fallas para componentes electrónicos típicos son tan pequeños, existen diversas unidades que se utilizan con frecuencia para expresarlos; porcentaje de fallas cada 10^3 horas, fallas cada 10^6 horas y fallas cada 10^9 horas. Esta última unidad se conoce como un *FIT*:

FIT

$$1 \text{ FIT} = 1 \text{ falla}/(10^9 \text{ horas})$$

En nuestro ejemplo inmediato anterior, diríamos que $\lambda_{\text{microprocesador}} = 800$ FIT.

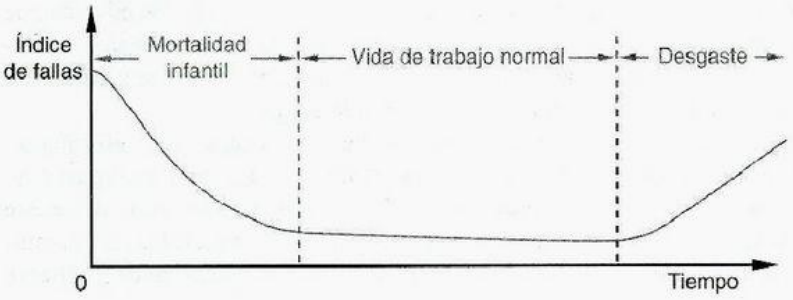


Figura 11-5
La "curva de tina de baño" para índices de fallas de componentes electrónicos.

El índice de fallas de un componente electrónico típico es una función del tiempo. Como se muestra en la figura 11-5, un componente típico tiene un alto índice de fallas al principio de su vida útil, tiempo en el cual la mayor parte de los defectos de fabricación se hacen visibles en forma espontánea; las fallas durante este periodo se conocen como *mortalidad infantil*. La mortalidad infantil es la razón por la cual los fabricantes de equipo de alta calidad realizan la *prueba de quemado (burn-in test)*, que también se conoce como prueba de calentamiento o prueba de operación previa al embarque: se hace funcionar el equipo de 8 a 168 horas antes de enviarlo al cliente. Con el calentamiento, la mayor parte de la mortalidad infantil se presenta en la fábrica en vez de ocurrir en las instalaciones de los clientes. Aun sin un periodo minucioso de calentamiento, los aparentemente mezquinos 90 días de garantía que ofrecen la mayoría de fabricantes de equipo electrónico cubren la mayor parte de las fallas que se presentan en los primeros *años* de operación del equipo (aunque si llega a fallar en el día nonagésimo primero, ¡eso es ya mala suerte!). Esto es bastante diferente de lo que ocurre con un automóvil u otra pieza de equipo mecánico, donde el "desgaste natural" incrementa el índice de fallas como una función del tiempo.

mortalidad infantil

prueba de quemado o calentamiento

Una vez que un componente electrónico ha pasado con éxito la prueba de quemado o calentamiento, se puede esperar que su índice de fallas sea bastante constante. Dependiendo del componente, pueden existir mecanismos de *desgaste* que se presentan posteriormente, durante la vida útil del componente, aumentando su índice de fallas. Anteriormente, los tubos de vacío se desgastaban después de varias miles de horas de operación porque sus filamentos se deterioraban debido al esfuerzo térmico. En la actualidad, la mayoría del equipo electrónico llega a ser obsoleto antes de que sus componentes de estado sólido comiencen a experimentar fallas por envejecimiento. Por ejemplo, aun cuando hayan pasado cerca de 25 años desde que comenzó el uso extendido de los dspositivos EPROM, muchos de los cuales garantizaban el almacenamiento de datos durante 10 años, no hemos visto una sucesión de fallas de equipo ocasionadas por la pérdida de sus bits. (¿Sabe de alguien que tenga una PC o VCR con más de 10 años de antigüedad?)

desgaste

De este modo, en la práctica, las fases de mortalidad infantil y desgaste en el tiempo de vida de un componente electrónico se ignoran, y la confiabilidad se calcula asumiendo que el índice de fallas es constante durante la vida de trabajo normal del equipo electrónico. Esta suposición, que nos dice que una falla es igualmente probable en cualquier momento de la vida útil del componente, nos permite utilizar un modelo matemático simplificado para predecir la confiabilidad del sistema, como lo mostraremos más adelante en esta sección.

Existen algunos otros factores que pueden afectar los índices de fallas del componente, incluyendo temperatura, humedad, golpes, vibraciones y variaciones cíclicas o intermitentes en la energía. El factor más importante para los CI es la temperatura. Muchos mecanismos de fallas de CI involucran reacciones químicas entre el chip y alguna

otra clase de contaminante, y éstas se aceleran conforme aumenta la temperatura. Del mismo modo, el exceso de tensión eléctrica en un transistor, que provoca un calentamiento excesivo y finalmente lo destruye, es peor si la temperatura del dispositivo es alta desde un principio. Tanto la evidencia empírica como la teórica apoyan la siguiente regla práctica ampliamente utilizada:

- El índice de fallas para un CI se duplica aproximadamente por cada 10°C de aumento en la temperatura.

Esta regla es cierta en mayor o menor grado para la mayoría de otras partes eléctricas.

Cabe indicar que la temperatura de interés en la regla anterior es la temperatura *interna* del CI, no la temperatura ambiente del aire que lo rodea. Un componente que acapare la potencia en un sistema que no tiene enfriamiento por ventilación forzada puede tener una temperatura interna del orden de 40–50°C más elevada que la temperatura ambiente. Un ventilador bien colocado puede reducir el aumento de temperatura a 10-20°C, reduciendo el índice de fallas del componente hasta por un factor de 10.

11.3.2 Confiabilidad y tiempo medio entre fallas

Para componentes con un índice de fallas constante λ, puede demostrarse que la confiabilidad es una función exponencial del tiempo:

$$R(t) = e^{-\lambda t}$$

La curva de confiabilidad en la figura 11-4 es una función de esta clase; da la casualidad de que emplea el valor $\lambda = 1$ falla/año.

tiempo medio entre fallas (MTBF)

Otra medida de la confiabilidad de un componente o sistema es el *tiempo medio entre fallas (MTBF, mean time between failures)*, que es el tiempo promedio que necesita un componente para empezar a fallar. Para los componentes que tienen un índice constante de fallas λ, puede demostrarse que MTBF es simplemente el recíproco de λ:

$$\text{MTBF} = 1/\lambda$$

11.3.3 Confiabilidad del sistema

Supongamos que construimos un sistema con m componentes, cada uno con un índice de fallas diferente, $\lambda_1, \lambda_2, \ldots, \lambda_m$. Supongamos que para que el sistema funcione apropiadamente, *todas* sus componentes deben funcionar en la forma adecuada. La teoría de básica de probabilidad nos indica que la confiabilidad del sistema está dada entonces por la fórmula

$$\begin{aligned} R_{\text{sis}}(t) &= R_1(t) \cdot R_2(t) \cdot \cdots \cdot R_m(t) \\ &= e^{-\lambda_1 t} \cdot e^{-\lambda_{2(1)} t} \cdot \cdots \cdot e^{-\lambda_m t} \\ &= e^{-(\lambda_1 + \lambda_2 + \cdots + \lambda_m)t} \\ &= e^{-\lambda_{\text{sis}} t} \end{aligned}$$

donde

$$\lambda_{\text{sis}} = \lambda_1 + \lambda_2 + \cdots + \lambda_m$$

Por tanto, la confiabilidad del sistema es también una función exponencial, que utiliza un índice de falla compuesto λ_{sistema} que es la suma de los índices de falla del componente individual.

La suposición de índice de fallas constante facilita la determinación de la confiabilidad de un sistema: solamente agregue los índices de fallas de los componentes individuales para obtener el índice de fallas del sistema. Los índices de fallas de componentes individuales pueden obtenerse de los fabricantes, de los manuales de confiabilidad o de los estándares de las compañías.

Por ejemplo, un fragmento del estándar de una compañía se exhibe en la tabla 11-1. Puesto que los índices de fallas son solamente estimados, esta compañía simplifica la labor del diseñador al considerar solamente categorías amplias de componentes, más que proporcionar índices de fallas "exactas" para cada componente. Otras compañías emplean listas más detalladas, algunos sistemas CAD mantienen una base de datos de índices de fallas de componentes, de manera que un índice de falla compuesto del circuito puede calcularse automáticamente a partir de su lista de partes.

Supongamos que vamos a construir un sistema en una tarjeta sencilla con un microprocesador VLSI, 16 chips de memoria y otros chips LSI, 2 chips SSI, 4 chips MSI, 10 resistores, 24 capacitores de desacoplamiento y conectores de 50 terminales. Aplicando los números de la tabla 11-1, podemos calcular el índice de fallas compuesto,

$$\lambda_{sis} = 1000 + 16 \cdot 250 + 2 \cdot 90 + 4 \cdot 160 + 10 \cdot 10 + 24 \cdot 15 + 50 \cdot 10 + 500 \text{ FITs}$$
$$= 7280 \text{ fallas} / 10^9 \text{ horas}$$

El MTBF de nuestro sistema de una sola tarjeta es $1/\lambda$, o aproximadamente 15.6 años. En efecto, los circuitos pequeños en realidad pueden ser así de confiables. Por supuesto, si incluimos una fuente de alimentación típica, con un índice de fallas de aproximadamente 10,000 FIT, el MTBF se reduce a más de la mitad.

Tabla 11-1
Tasas de falla de componentes típicas a 55 °C.

Componente	*Índice de fallas (FIT)*
CI SSI	90
CI MSI	160
CI LSI	250
Microprocesador VLSI	500
Resistor o resistencia	10
Capacitor de desacoplamiento	15
Conector (por terminal)	10
Tarjeta de circuito impreso	1000

11.4 Líneas de transmisión, reflexiones y terminaciones

Nada ocurre de manera instantánea, especialmente donde se encuentran involucrados los circuitos digitales. En particular, considere el hecho de que el retardo de propagación de "la velocidad de la luz" de las señales eléctricas en el *alambre* se encuentra en el orden de los 1.5 – 2 ns por pie de longitud (el retardo exacto depende de las características del alambre). Cuando los retardos en el alambre son semejantes a los tiempos de transición de las señales que conducen, debemos tratar a los alambres no como conductores perfectos con retardo cero, sino como "líneas de transmisión" que son en realidad.

El comportamiento de línea de transmisión incluye cambios en la señal que podrían no ser predecibles mediante un análisis DC del funcionamiento del circuito. Los cambios más significativos se presentan durante un intervalo de aproximadamente $2T$ después de que una salida cambia de estado, donde T es el retardo desde la salida hasta el extremo lejano del alambre que controla.

Así también, su comportamiento por lo regular no afecta el funcionamiento lógico de los dispositivos cuyos tiempos de transición de señal y retardos de propagación son mucho mayores que $2T$. De este modo, el comportamiento se considera típicamente sólo para conexiones 74LS sobre poco más de medio metro de longitud, para conexiones de 74AS, 74AC y 74ACT sobre 30cm de longitud y para conexiones 74FCT y ECL sobre 7.5 a 15cm. Naturalmente, se puede salir impune con conexiones largas dependiendo de la aplicación, pero también existen situaciones donde la teoría de la línea de transmisión debe aplicarse incluso a conexiones más cortas. Incluso algunos chips VLSI consiguen velocidades suficientemente altas como para que el comportamiento de línea de transmisión deba ser considerado en el diseño interno del chip.

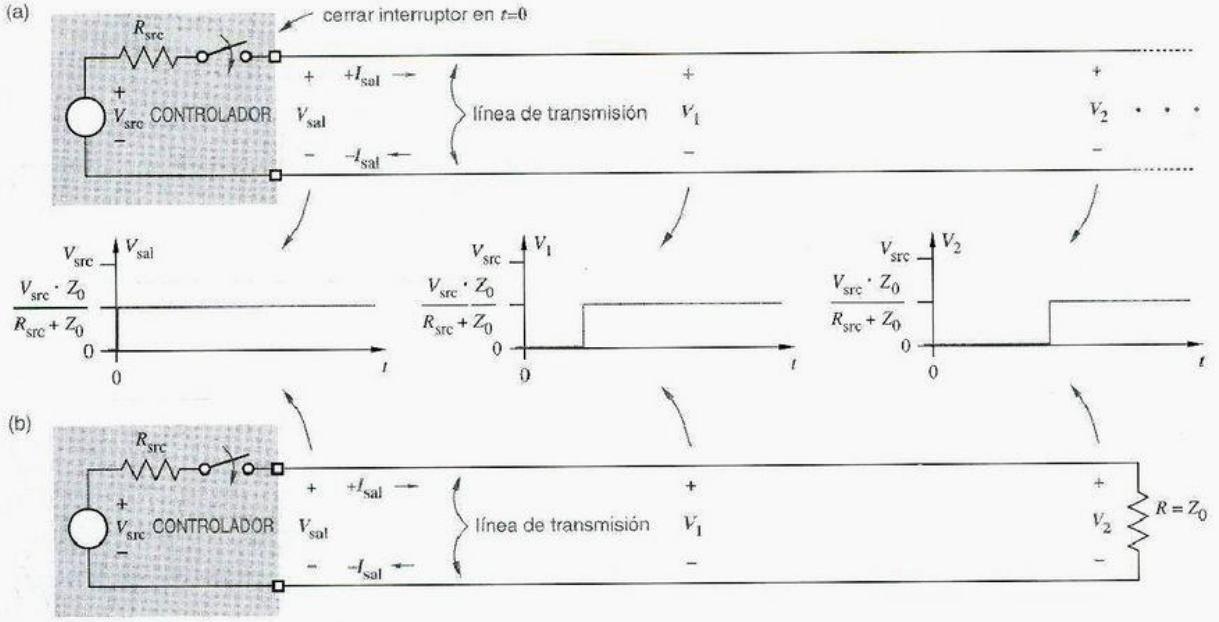
11.4.1 Teoría básica de línea de transmisión

línea de transmisión

Dos conductores en paralelo constituyen la *línea de transmisión* más simple. Considere un par de conductores con longitud infinita, como se muestra en la figura 11-6(a). Si colocamos de manera instantánea una fuente de voltaje a través del par, cierta corriente fluirá para crear una onda de voltaje que viaje a lo largo del par. La razón del voltaje con respecto a la corriente, V_{sal}/I_{sal}, depende de las características físicas de los conductores y se denomina la *impedancia característica* Z_0 del par conductor.

impedancia característica Z_0

Figura 11-6 Líneas de transmisión: (a) con longitud infinita; (b) con longitud finita, terminada con impedancia característica.



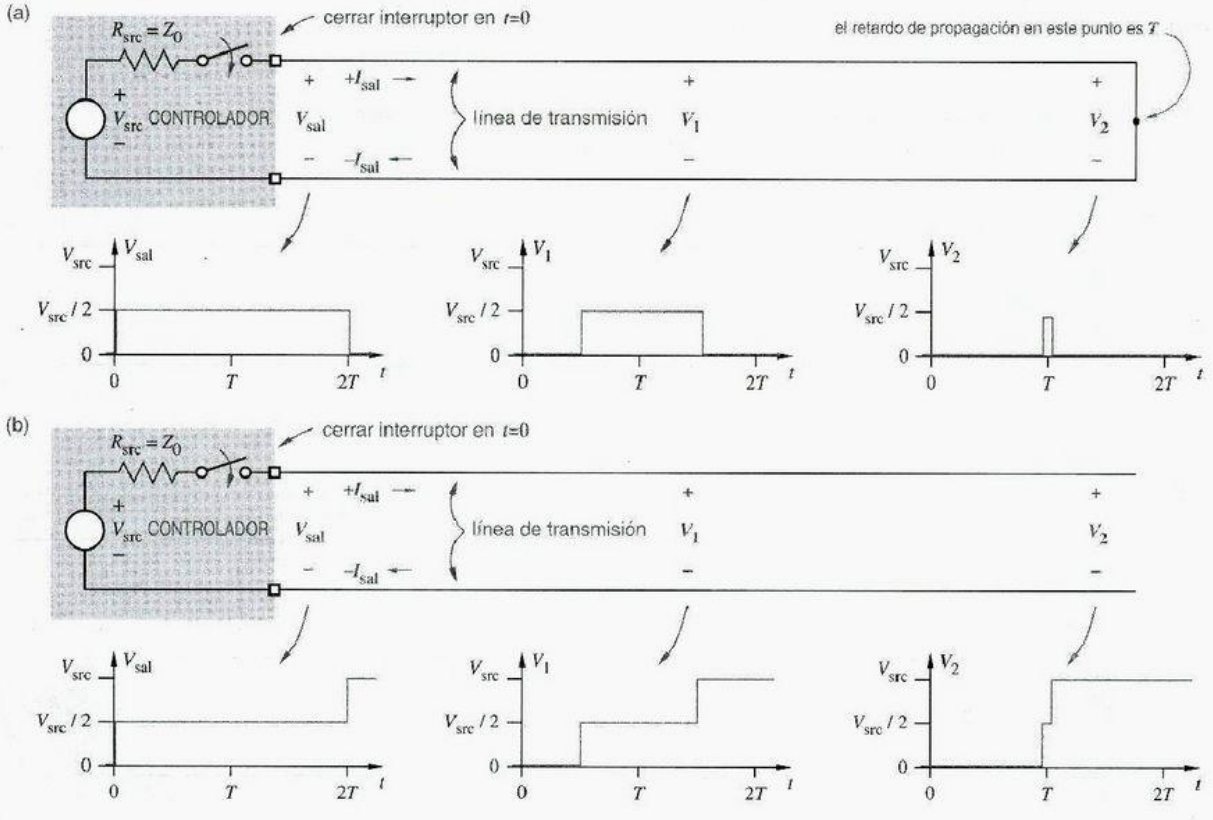
La magnitud de V_{sal} es determinada mediante la consideración de la combinación en serie de R_{src} y Z_0 como un divisor de voltaje, de manera que

$$V_{sal} = V_{src} \cdot \frac{Z_0}{R_{src} + Z_0}$$

Por supuesto, no tenemos ningún conductor infinitamente largo. Sin embargo, supongamos que tenemos un par de conductores de un metro y medio de longitud, y que colocamos una resistencia igual a Z_0 a través del extremo final. Si colocamos instantáneamente una fuente de voltaje a través del par como se ilustra en (b), la misma corriente fluye indefinidamente como en el caso de longitud infinita. De este modo, cuando una línea es terminada por su impedancia característica, ya no necesitamos considerar los efectos de línea de transmisión.

La situación es diferente para una línea de transmisión de longitud finita que no es terminada en su impedancia característica. Un caso extremo, en el cual el extremo final se encuentra en cortocircuito, se muestra en la figura 11-7(a). Por simplicidad, suponemos que $R_{src} = Z_0$ en este ejemplo. Inicialmente, todo lo que el controlador mira es la impedancia característica de la línea, y una onda de voltaje con amplitud $V_{src}/2$ afortunadamente se propaga hacia debajo de la línea. Sin embargo, cuando la onda incide en el extremo final al tiempo T, mira el cortocircuito. A fin de satisfacer las leyes de Kirchhoff,

Figura 11-7 Líneas de transmisión: (a) en cortocircuito; (b) en circuito abierto.



una onda de voltaje de polaridad *inversa* se propaga de vuelta hacia abajo de la línea, cancelando la onda original. El extremo final ha *reflejado* la onda original, y el controlador observa el cortocircuito al tiempo $2T$. *reflexión*

Otro caso extremo, en el cual el extremo final se encuentra en circuito abierto, se ilustra en (b). Todo se inicia como antes. Sin embargo, cuando la onda de voltaje inicial incide en el extremo final, la corriente no tiene adónde ir, y entonces una onda de voltaje de la *misma* polaridad se propaga de regreso hacia arriba de la línea, agregándose al voltaje original. Cuando la onda reflejada alcanza el controlador en el tiempo $2T$, el voltaje en todas partes es V_{src}, y no ocurre nada más.

En el caso general, la amplitud de la onda reflejada al final de la línea de transmisión es determinada por el *coeficiente de reflexión*, ρ (rho). El valor de ρ depende de Z_0 y de Z_{term}, la *impedancia de terminación* que aparece al final de la línea: *coeficiente de reflexión, ρ* *impedancia de terminación*

$$\rho = \frac{Z_{term} - Z_0}{Z_{term} + Z_0}$$

Cuando una onda de voltaje con amplitud V_{onda} llega al final de una línea de transmisión, una onda con amplitud $\rho \cdot V_{onda}$ se refleja. Advierta que tres casos simples satisfacen nuestra discusión anterior:

$Z_{term} = Z_0$ Cuando una línea de transmisión es terminada en su impedancia característica, el coeficiente de reflexión es 0.

$Z_{term} = 0$ El coeficiente de reflexión de una línea en cortocircuito es –1, produciendo una reflexión de igual magnitud pero polaridad opuesta.

$Z_{term} = \infty$ El coeficiente de reflexión de una línea en circuito abierto es +1, produciendo una reflexión de igual magnitud y la misma polaridad.

Si la impedancia fuente (R_{src}) no es igual a Z_0, entonces las reflexiones se presentan en el extremo cercano de la línea así como también en el extremo final. Cada extremo de la línea tiene su propio valor de ρ. Se aplica el *principio de superposición*, de manera que el voltaje en cualquier punto sobre la línea y en cualquier instante en el tiempo sea la suma de esa condición inicial del punto y todas las ondas que hayan pasado hasta entonces. Se dice que el final de una línea de transmisión *coincide* si está terminada con su impedancia característica. *principio de superposición* *coincidencia de línea de transmisión*

La figura 11-8 ilustra el comportamiento de una línea de transmisión que no coincide en ningún extremo. (Note que la fuente de voltaje ideal del controlador V_{src} se considera que tiene una resistencia de 0 Ω para este análisis.) Las reflexiones se presentan en ambos extremos de la línea, con ondas más y más pequeñas reflejándose hacia atrás y hacia delante. El voltaje en todas partes sobre la línea se aproxima de manera asintótica a 0.9 V_{src}, el valor que sería predicho por un análisis "DC" del circuito utilizando la ley de Ohm.

11.4.2 Interconexiones de señal lógica como líneas de transmisión

Así, ¿cómo afecta la teoría de la línea de transmisión a las señales lógicas? Consideremos el caso de una salida 74HC CMOS controlando una entrada CMOS al final de una línea de transmisión que consiste de una línea de señal y tierra, como se ilustra en la figura 11-9. Un transistor de salida "encendido" de canal-*p* o canal-*n* de la familia HCT

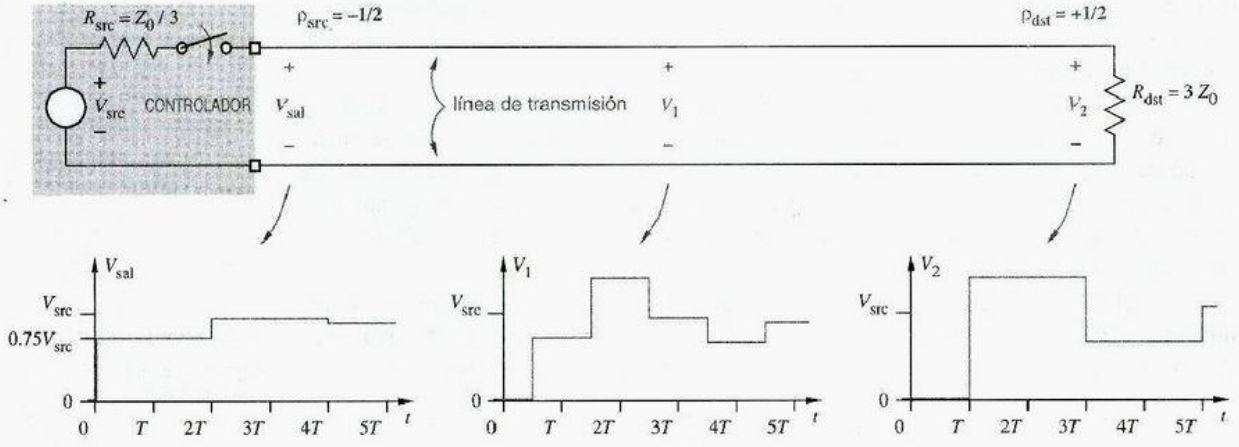
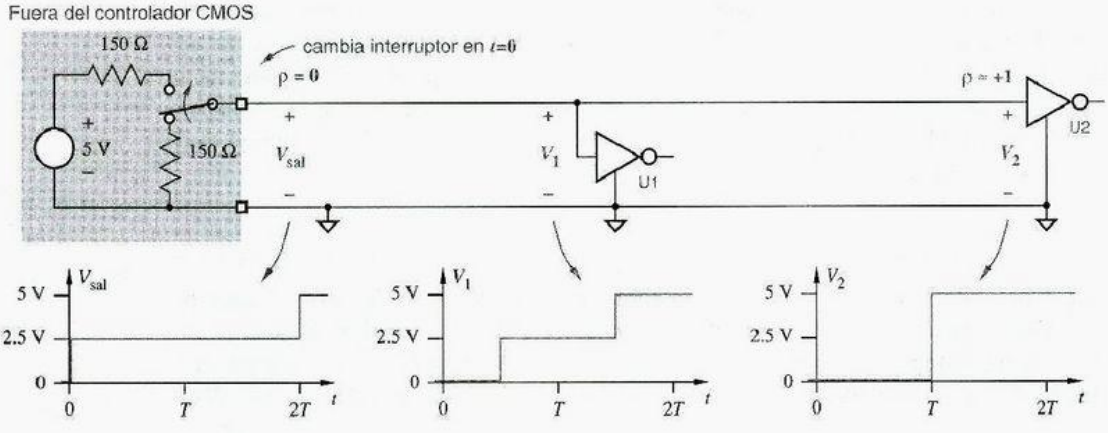


Figura 11-8 Una línea de transmisión que no coincide en ningún extremo.

tiene una resistencia de aproximadamente 100-200 Ω; por simplicidad supongamos 150 Ω ya sea en el estado ALTO o BAJO. La impedancia característica de una pista típica de PCB con respecto a tierra está en el orden de 100-150; supongamos que sea de 150 Ω por conveniencia, de modo que el coeficiente de reflexión en el extremo de control de la línea sea 0. Una entrada CMOS típica tiene una impedancia por arriba de 1 MΩ, de manera que el coeficiente de reflexión en el extremo de recepción sea efectivamente +1.

En la figura, consideramos el caso de una transición BAJO a ALTO. Cuando el controlador CMOS conmuta de BAJO a ALTO, la fuente de 5 V en el controlador mira la resistencia de 150 Ω del controlador en serie con la Z_0 de 150 Ω de la línea, de modo que una onda de 2.5 V se propaga hacia debajo de la línea. Después de un tiempo T, esta onda alcanza la compuerta de recepción U2 en el extremo final y es reflejada. Después del tiempo $2T$, la onda reflejada alcanza el extremo de envío y es absorbida sin reflexión, debido a que $\rho = 0$ en ese extremo.

Figura 11-9 Reflexiones en una línea de señal lógica que cambia de BAJO a ALTO.



Todo funciona de maravilla en lo que a la compuerta de recepción U2 se refiere: mira una transición instantánea de 0 a 5 V al tiempo T después de que el controlador conmuta. Sin embargo, considere la forma de onda vista por otra compuerta de recepción U1 ubicada a medio camino entre el controlador y U2. Como se muestra en la figura 11-9, U1 ve una entrada de solamente 2 V para un intervalo T. Una compuerta de recepción ubicada más cerca del controlador vería este voltaje de entrada aún más grande. Éste es un problema, porque 2.5 V está bien para el umbral de conmutación para entradas CMOS de 5 V. Si este voltaje de entrada fuera mantenido por mucho tiempo, la salida de U2 podría oscilar o producir un voltaje no lógico.

Cualquier conexión de señal lógica, sin importar qué tan larga o corta sea, es una línea de transmisión. Sin embargo, el análisis anterior está bastante idealizado. En la práctica, el comportamiento de línea de transmisión no causa problemas si T es algo más corto que los tiempos de transición de señales lógicas o los retardos de propagación de las compuertas lógicas. En un caso así, las reflexiones por lo regular se establecen antes de que los receptores sobre la línea tengan una oportunidad de advertir su presencia.

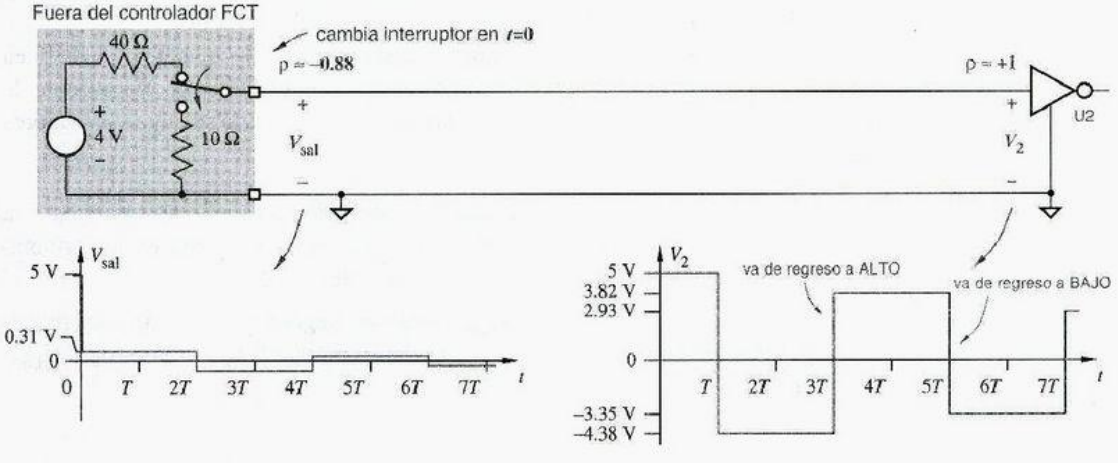
Un caso algo más desagradable se presenta cuando una salida de alto desempeño cambia de ALTO a BAJO. Por ejemplo, como se muestra en la figura 11-10, en el estado BAJO una salida FCT CMOS parece aproximadamente como un resistor de 10 Ω a tierra. De esta forma, el coeficiente de reflexión en el extremo de envío es de aproximadamente –0.88. Cuando la salida se conmuta primero a BAJO, ve un divisor de voltaje que se compone del resistor de 10 Ω en serie con la impedancia de línea de 150 Ω. Puesto que la línea estaba originalmente a 5 V, el nuevo voltaje en la salida del divisor debe ser

$$V_{sal} = \frac{10\Omega}{150\Omega + 10\Omega} \cdot 5\ \text{V} = 0.31\ \text{V}$$

Por tanto, una onda de voltaje (0.31 – 5.0) = –4.69 V se propaga hacia abajo de la línea. Cuando esta onda alcanza el extremo de recepción al tiempo T, produce una reflexión de igual signo y magnitud (puesto que ρ = 1). Así, el voltaje en el extremo de recepción es ahora $V2 = 5.0 - 4.69 - 4.69 = -4.38$ V, ¡un voltaje negativo! Esto se conoce como *transitorio de bajo voltaje*.

transitorio de bajo voltaje

Figura 11-10 Reflexiones en una línea de señal lógica que cambia de ALTO a BAJO.



Cuando la reflexión de –4.69 V retorna al extremo de envío al tiempo 2*T*, ocurre aún otra reflexión. Esta vez, puesto que ρ es negativa, la onda reflejada tiene una amplitud positiva, equivalente a $-0.88 \cdot -4.69 = +4.10$ V. El voltaje de salida está ahora, modificado por la suma de la reflexión entrante y la nueva reflexión saliente: $V_{sal} = 0.31 - 4.49 + 4.10 = -0.08$ V. Esto no es problema. Sin embargo, cuando la reflexión de +4.10 V alcanza el extremo de recepción al tiempo 3*T*, la reflexión positiva consiguiente produce $V2 = -4.38 + 4.10 + 4.10 = 3.82$ V, ¡volviendo a ser positiva otra vez!

Como se sugiere en la figura, la reflexión continúa, con el voltaje aproximándose de manera asintótica a 0 V tanto para el extremo de envío como para el de recepción, valor que se pronosticaría con un análisis DC del circuito. Este patrón de oscilación es conocido como *sobreoscilación.*

sobreoscilación

La gran magnitud de la sobreoscilación en el extremo de recepción puede ser un problema, puesto que V_2 no establece un intervalo que sea menor que el voltaje de entrada en estado BAJO (0.8 V) hasta el tiempo 2*T*. De esta forma el retardo de propagación efectivo del circuito ha sido incrementado por muchas veces el retardo del alambre. Peor aún, si la señal es un reloj, se detectarán flancos extra en las transiciones de la sobreoscilación, temporizando falsamente los flip-flops en el extremo de recepción.

Una vez más, el comportamiento de línea de transmisión que hemos descrito no ocasiona problemas si *T* es mucho más corto que los tiempos de transición de las señales lógicas o los retardos de propagación de las compuertas lógicas. Además, estructuras de entrada TTL y muchas estructuras de entrada CMOS incluyen *diodos de bloqueo (fijadores de nivel),* que son los diodos polarizados inversamente de cada entrada a tierra que mostramos en la figura 3-75. Puesto que estos diodos cambian a impedancia de entrada del receptor de muy alta a muy baja para voltajes negativos, limitan la excursión negativa al tiempo *T* aproximadamente a 1 V. Esto reduce la reflexión de regreso al extremo de envío, la cual a su vez disminuye la excursión al tiempo 3*T* a menos de 1 V. Las entradas de algunos dispositivos también tienen diodos conectados a V_{CC} para manejar el sobreimpulso que se presenta cuando los controladores de baja impedancia efectúan una transición de BAJO a ALTO.

diodos de bloqueo (fijadores de nivel)

11.4.3 Terminaciones de señal lógica

Las reflexiones pueden ser eliminadas mediante líneas de transmisión de terminación en sus impedancias características. Se emplean por lo regular dos métodos.

En una *terminación de Thévenin,* un par de resistores se colocan al final de la línea como se muestra en la figura 11-11(a). Esta terminación tiene el circuito equivalente de Thévenin que se ilustra en (b). Se consideran varios factores cuando se eligen los valores de los resistores:

terminación de Thévenin

1. La resistencia equivalente de Thévenin R_{Thev} debería ser igual o aproximarse a la impedancia característica de línea, Z_0. La resistencia de Thévenin es la combinación en paralelo de *R1* y *R2*, es decir, $(R1 \cdot R2)/(R1 + R2)$.
2. El voltaje equivalente de Thévenin V_{Thev} puede ser elegido para optimizar los requerimientos de corriente de sumidero y fuente de la compuerta de control. Aquí, V_{Thev} es igual a $V_{CC} \cdot R2/(R1 + R2)$.

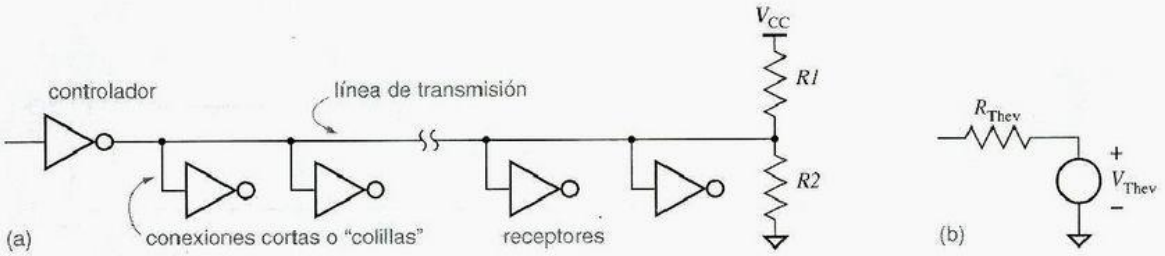


Figura 11-11 Terminación de Thévenin: (a) circuito; (b) circuito equivalente para terminación.

- Por ejemplo, si la compuerta de control puede suministrar y drenar iguales cantidades de corriente tanto en el estado ALTO como en el BAJO (como en un CMOS estándar, no compatible con TTL), entonces V_{Thev} puede encontrarse a la mitad del camino entre V_{OL} y V_{OH}.
- Por otra parte, si la compuerta de control puede drenar más corriente de la que puede suministrar (como en TTL y TTL compatible CMOS), entonces se elige un valor más alto de V_{Thev} para disminuir los requerimientos de suministro de corriente del controlador en el estado ALTO, a expensas de aumentar sus requerimientos de drenado de corriente en el estado BAJO.

3. Para buses de tres estados, el voltaje equivalente de Thévenin V_{Thev} puede ser escogido para producir un nivel lógico válido cuando el bus no se encuentre siendo controlado. Es especialmente importante en este caso elegir un voltaje que no se encuentre cerca del umbral de conmutación de los receptores, a fin de que no conduzcan corriente excesiva u oscilen.
4. Los valores finales del resistor se eligen para satisfacer valores de componentes estándar (por ejemplo, 150, 220, 270, 330, 390 y 470 ohms).

Por ejemplo, una terminación "estándar" en algunas aplicaciones TTL tiene $R1 = 220\ \Omega$ y $R2 = 330\ \Omega$, produciendo $R_{Thev} = 132\ \Omega$ y $V_{Thev} = 3.0$ V. En el estado BAJO, un controlador debe consumir $(3.0\text{ V})/(132\ \Omega) = 22.7$ mA, y en el estado ALTO, no es necesaria ninguna corriente para mantener un nivel lógico válido.

Con una terminación Thévenin bien acoplada al extremo de una línea de transmisión, se presentará poca o nula reflexión. El punto débil de esta terminación es que consume potencia de DC todo el tiempo, y se requieren corrientes de controlador relativamente altas. Una *terminación de fuente* (en ocasiones denominada una *terminación en serie*) supera estos problemas.

terminación de fuente
terminación de series

Como se ilustra en la figura 11-12, una resistencia igual a Z_0 menos R_d, la típica impedancia de salida del controlador, se coloca en serie con la salida, físicamente cerrada para la salida misma. Las formas de onda de tiempo para esta terminación se ilustran en la

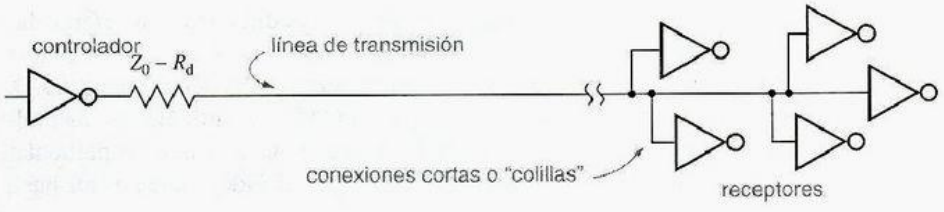


Figura 11-12
Terminación de fuente.

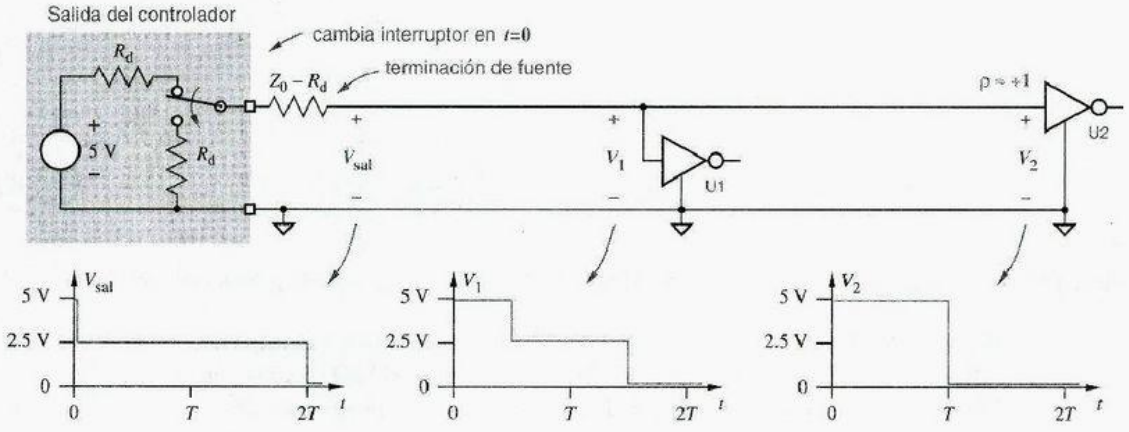


Figura 11-13 Reflexiones en una línea terminada de fuente que cambia de BAJO a ALTO.

figura 11-13. En una transición de V_{CC} a 0 (o viceversa), el controlador mira una resistencia en serie de $2Z_0$, el resistor en serie más la línea misma. La mitad del cambio de voltaje se desarrolla a través del resistor en serie y la otra mitad a través de la línea misma. De este modo, después de un retardo de transmisión T en un sentido, el receptor del extremo final U2 observa un cambio de voltaje de $V_{CC}/2$. Suponiendo un coeficiente de reflexión de 1, este valor total es inmediatamente reflejado de regreso hacia el controlador, conduciendo el voltaje a U2 todo el recorrido hasta V_{CC}. Cuando esta reflexión alcanza al controlador, es absorbida en la resistencia en serie coincidente Z_0, y la línea se encuentra estable.

La terminación de la fuente funciona bien para controladores cuyas impedancias de salida en los estados BAJO y ALTO sean aproximadamente iguales para propósitos prácticos (por ejemplo, CMOS). Los diseños típicos hacen uso de resistores en serie de 15-40 Ω para líneas de transmisión de 50 a 100 Ω.

Esta misma terminación de fuente es indeseable si cualquier compuerta receptora, tal como U1 en la figura, se coloca bajo la línea de transmisión. En una transición, estas compuertas "medias" ven una transición inicial de solamente $V_{CC}/2$; ellos observan la transición completa solamente después de que va la reflexión desde el extremo. Puesto que $V_{CC}/2$ está probablemente cerca del umbral de conmutación de las compuertas receptoras éste no es un buen lugar para estar por mucho tiempo. Por lo tanto, la terminación de fuente se utiliza normalmente sólo cuando un PCB está físicamente extendido en una manera similar a la figura 11-12, con una línea relativamente larga controlando una carga o unas cuantas que se encuentran estrechamente agrupadas.

Referencias

Las herramientas CAD son sólo herramientas, y usted no podrá encontrar referencias académicas acerca de cómo utilizarlas. Sin embargo, *IEEE Design & Test of Computers (D&T)* es una excelente fuente de información actual acerca de las herramientas CAD. La mejor fuente de información para una herramienta CAD en particular es, naturalmente, la documentación del distribuidor. Cuando se utiliza una herramienta particular de manera *muy* extendida, puede aparecer una guía tutorial independiente; un buen

ejemplo es *SPICE: A Guide to Circuit Simulation and Analysis Using Pspice®*, por Paul W. Tuinenga (Prentice Hall, 1988).

Los lenguajes de transferencia de registro se discuten en la mayoría de los libros de diseño computacional, incluyendo *Computer Architecture and Organization*, segunda edición, por John P. Hayes (McGraw-Hill, 1988). La revista *IEEE Computer* publicó un número especial sobre lenguajes de descripción de hardware en febrero de 1985, y *D&T* publicó un número especial sobre VHDL en el mes de abril de 1986.

El fidedigno tratamiento del diseño para pruebas en el contexto del diseño lógico se encuentra en *Logic Design Principles* (Prentice Hall, 1986) de Edward J. Mccluskey. Un tratamiento a fondo del tema desde el punto de vista de ingenieros de fabricación y prueba es *Digital Test Engineering* por J. Max Cortner (Wiley, 1987). Una vez más, *D&T* es una excelente fuente de información actual. En el número especial de agosto de 1989 acerca de los métodos de autoprueba integrados, un artículo de Kenneth P. Parker, "The Impact of Boundary Scan on Board Test", ofrece una perspectiva muy práctica sobre la creciente importancia de los métodos de exploración en el diseño al nivel de tarjeta.

Los antecedentes matemáticos requeridos y métodos generales para analizar la confiabilidad del sistema puede encontrarse en un práctico libro cuyo autor es J. C. Cluley, *Electronic Equipment Reliability* (Halsted Press/Wiley, 1974). Para ver cómo fallan realmente los chips, y cómo pueden ser precedidos sus índices de fallas en menos de una vida (de usted o de un chip), debería leer *Failure Mechanisms in Semiconductor Devices* por E. A. Amerasekera y D. S. Campbell (Wiley, 1987).

Una excelente introducción a la teoría y práctica de líneas de transmisión desde el punto de vista de un diseñador digital puede hallarse en *Digital Design with Standard MSI and LSI*, segunda edición, por Thomas R. Blakeslee (Wiley, 1979).

Una lectura altamente recomendada y un tratamiento fidedigno de los aspectos analógicos del diseño digital en que me gusta insistir es *High-Speed Digital Design* por Howard W. Johnson y Martin Graham (Prentice Hall, 1993). Aunque el subtítulo del libro es *A Handbook of Black Magic*, los autores recorren un largo camino hacia la desmitificación del comportamiento analógico con el que todo diseñador de sistemas digitales de alta velocidad debe arreglárselas.

ÍNDICE

Nota: Los números de página para las referencias a definiciones se encuentran en **negritas**.

&*%$#@!

& (ABEL AND), **251**
! (ABEL NOT), **251**, 252, 363
(ABEL OR), **251**
!$ (ABEL XNOR), 251
$ (ABEL XOR), **251**
@ALTERNATE, comando, ABEL, **251**
^b, prefijo binario, ABEL, **259**
@CARRY, comando, ABEL, **444**
:, en nombre del bus, 327
⌈ ⌉ (función techo), **53**
^h, prefijo hexadecimal, ABEL, **259**
ΔI_{CC}, 139
_L, sufijo, **318**, 354, 356
<, operador, ABEL, **259**
/, prefijo, ABEL, **250**
:=, operador, ABEL, **628**
:>., operador, ABEL, **628**
+, operador, ABEL, **705**
=, operador, ABEL, **252**, 254
==, operador, ABEL, **259**
->, operador, ABEL, **257**
>, operador, ABEL, **259**
>=, operador, ABEL, **259**
<=., operador, ABEL, **259**
&, operador de concatenación VHDL, **276**, 502, 743
<, operador, VHDL, **287**
<=, operador, VHDL, **287**
/=, operador, VHDL, **287, 427**
:=, operador, VHDL, **290**
=, operador, VHDL, **287, 427**
>, operador, VHDL, **287**
>=, operador, VHDL, **287**
* sufijo, **552, 554**
.AP, sufijo, ABEL, **628**
.AR, sufijo, ABEL, **628**
.CLK., sufijo, ABEL, **628**
comando @DCSET, ABEL, **260**
.FB, sufijo, ABEL, **633**
.OE, sufijo, ABEL, **391, 628**
operador !=, ABEL, 259
.PIN, sufijo, ABEL, **633**
prefijo ', como en ', **139, 354**
.Q, sufijo, ABEL, **633**
símbolo .C., ABEL, **638**
símbolo OR exclusivo ⊕, 305, **411**
.SP, sufijo, ABEL, **628**
.SR, sufijo, ABEL, **628**
.X., símbolo, ABEL, **252**
^ (vee) 197

0

0, 3
0 lógico, 8
0 y 1, 3, 7, 8, 25, 80, 86, 195, 317, 836
1, 3
1 lógico, 8
1, conjunto, **300**
1076, paquetes aritméticos IEEE, 281
1164, paquete lógico estándar IEEE, 273, 276, 277, 281
128V64, 872
16V8, 733, 798, 802
16V8C, 674, **685**
16V8R, **685**
16V8S, **685**, 687
1-bit, código de paridad de, **60**
1 fuera de 10, código, **51**
1 fuera de *m*, código, 351
1 fuera de *n*, código, **55**, 572
20V8, 486, **687**, 707
22V10, 486, **689**, 798, 802
2421, código, **50**
27128, **844**
27256, **844**
27512, **844**
2764, **844**
2 a 4, decodificador, 352, 354, 355
2^n a *n*, codificador, **376**
3 a 8, decodificador, 357, 359
4 a 16, decodificador, 359
4 a 2, codificador, 378
5 V, entradas con tolerancia de, 173, 174
5 V, salidas con tolerancia de, 174
54, partes de la serie, 137, 168
54, prefijo, 98, 135
5 a 32, decodificador, 359
74, CI de la serie, 312
74, partes de la serie, 135, 137

74, prefijo, 98, 135
74, TTL de la serie, **166**
7497, 789
74ACT74, **666**
74ALS TTL Schottky avanzado de baja potencia (Advanced Low-power Schottky), **167**
74ALS74, 770
74AS, Schottky avanzado (Advanced Schottky) TTL, **166**
74AS374, 773
74AS4374, **773**
74F, TTL rápido (Fast TTL), **167**
74F373, 769
74F374, 769
74F74, **666**, 769
74H, TTL de alta velocidad (High-speed TTL), **166**
74L, TTL de baja potencia (Low-power TTL), **166**
74LS, TTL Schottky de baja potencia (Low-power Schottky TTL), **166**, 669
74LS00, 168
74LS138, 336
74LS139, 336
74LS74, 542, 548, 627, 768, 769, 770
74LS86, 336
74S, TTL Schottky (Schottky TTL), **166**
74S174, 769
74S373, 769
74S374, 769
74S74, 769
74VHC1G00, 14
74x, prefijo 138, **354**
74x00, 137, 168, 329, 334
74x02, 329, 334
74x03, 329
74x04, 329, 334
74x08, 329, 334
74x10, 329, 334
74x109, **548, 666**
74x11, 329, 334
74x112, **666**
74x125, **387**
74x126, **387**
74x138, 137, 143, 335, **357**, 359, 361, 385, 405,415, 475, 477, 703—704
74x139, 335, 354, **355**, 359, 405
74x14, 329, 334
74x148, **378**, 472, 473, 475, 477
74x151, 335, **399**, 402, 469, 472, 473
74x153, 335, 391, **401**, 405
74x157, 335, **401**, 470, 478, 754
74x160, **699**
74x161, **699**
74x162, **699**
74x163, **696**, 696—704, 705, 720—723, 760
74x164, **714**, 723
74x166, **714**, 723
74x169, **703**
74x174, **672**
74x175, **671**
74x181, **439**, 439—440
74x182, 335, **441**
74x194, **714**—716, 732, 733, 754, 760
74x20, 329, 334
74x21, 329, 334
74x240, **390**
74x241, **390**, 760
74x245, **390**
74x251, **402**, 404
74x253, **402**
74x257, **402**
74x266, 329
74x27, 329, 334
74x280, 335, **413**, 415
74x283, 335, 422, 423, **436**—438, 754
74x299, 716
74x30, 329, 334
74x32, 329, 334
74x373, 535, **673**
74x374, 535, **672**, 704, 760
74x375, **666**
74x377, 673, 723, 754, 760, 762
74x381, 335, **440**
74x382, **440**
74x49, **372**
74x540, **390**
74x541, **387**, 415
74x682, 335, **424**, 426, 478
74x74, **666**
74x83, **436**
74x85, 422, 423—426
74x86, 329, 334, **412**, 420, 478, 732
8 entradas, codificador de prioridad de, 378
82S100, 340
8421, código, **50**
8 a 3, codificador, 376
8B10B, código, 56, 71, 74
9, complemento a, **38**
captura de ceros, **547, 548**
complemento a 10, **35**
conjunto 0, **300**
mapa de Karnaugh de 5 variables, **307, 574**
mapa de Karnaugh de 6 variables, **308**
prefijo 0x, 29
representación del complemento a 10, 50
serie 4000, CMOS, **135**
unos, captura de, **547**, 548
unos, máquina de conteo de, 580, 799, 819

A

a, frecuencia de eventos asincrónicos, **768**
ABEL, **249**
ABEL, 9, 16, 249—263
@ALTERNATE, comando, **251**
&(AND), **251**
^b, prefijo binario, **259**
@CARRY, comando, **444**
@DCSET, comando, **260**
^h, prefijo hexadecimal, **259**
!(NOT), **251**, 252, 363
!=, operador, **259**
:=, operador, **628**
+, operador, 705
<, operador, **259**
<=, operador, **259**
=, operador, **252**, 254
==, operador, **259**
—>, operador, **257**
>, operador, **259**
>=, operador, **259**
#(OR), **251**
/, prefijo, **250**
!$ (XNOR), **251**
$(XOR), **251**
.AP, sufijo, **628**
.AR, sufijo, **628**
.C., esquina de reloj, **638**

.CLK, sufijo, **628**
.FB, sufijo, **633**
.OE, sufijo, **391, 628**
.PIN, sufijo, **633**
.Q, sufijo, **633**
.SP,sufijo, **628**
.SR, sufijo, **628**
.X., símbolo, **252**
atributo, sufijo de, **391**, 628, 632
bloque de ecuaciones, **254**
buffer, palabra clave, **252**
cadena de caracteres, **250**
CASE, instrucción, 630
cd, palabra clave, **260**
com, palabra clave, **250**
combinación de entrada *sin importancia*, 260
comentarios, **250**
compilador, **249**, 563
conjunto, **258**, 629, 634
contadores, 705
declaración de dispositivo, **250**
diagrama de estado, 629—637
ecuación intermedia, **252**, 483
ecuaciones, **250**
ELSE, cláusula, **253, 630**
ENABLE, palabra clave, 252
end, instrucción, **252**
expresión constante, 362
expresión relacional, **258**
GOTO, instrucción, **630**
identificador, **249**
IF, instrucción, **630**
intervalo, **258**
invert, palabra clave, **252**
istype, palabra clave, **250**
lista de entrada, **257, 262**
lista de propiedad, istype, **250**
lista de salida, **257, 262**
módulo, instrucción, **249**
neg, palabra clave, 252
NOT, prefijo, **250**
operador de asigación no temporizado, =, 252, 254
operador de asignación temporizado, :=, **628**
operador de tabla de verdad no temporizado, ->, **257**
operador de tabla de verdad temporizado, :>, **628**
operadores relacionales, **258**
otras declaraciones, 250
pos, palabra clave, 252
precedencia, 251
precedencia de operadores, 251
procesador de lenguaje, **249**
reg, palabra clave, **628**
relación, **258**
retain, propiedad, **675**
sensibilidad al tamaño de letra, 249
state, 630
state_diagram, 629
tabla de verdad, **257**, 260, 628
terminal, declaraciones de, **250**
terminal, definiciones de, 363
test_vectors, **251, 262**
title, **250**
truth_table, palabra clave, **257**
valor de estado, **630**
variable de estado en el lado izquierdo, 630
equations, instrucción, **250**
variables de estado actual, **634**
variables de estado siguiente, **634**
vector de estado, **629**
vectores de prueba, **251**, 638
WHEN, instrucción, **253**
WITH, instrucción, **636**
abogados, 126, **340**
abstracción digital, 96, 195
acarreo, 32, **43**
acarreo cerca del extremo, **44**
acarreo de salida, 431
acarreo en grupo hacia delante, **441**
acarreo hacia delante, **434**
acción del diodo, 146
Active-CAD, xxiv
Active-HDL, xxiv
activo en estado alto, **317**
activo en estado bajo, **317**
afirmación, **317**
after, palabra clave, VHDL **295**
AHC CMOS avanzado de alta velocidad (Advanced High-speed CMOS), 137
AHCT, CMOS avanzado de alta velocidad compatible con TTL (Advanced High-speed CMOS, TTL compatible), 137
ajuste, **267**
Aldec, Inc., xxiv
alfa, partícula, 58
Alfke, Peter, 183, 784
álgebra binaria (álgebra de conmutación), 194, **195**-209, 298
distribución de la suma, 199
expresión ambigua, 198
ley asociativa, 198
ley conmutativa, 198
operador binario, **199**
teorema de combinación, 199
álgebra de conmutación usando los paréntesis, 198
algoritmo de consenso iterativo, 299
algoritmo de Quine-McCluskey, 236, 236—243
almohadillas de prueba, **904**
almuerzo, 3
alterado, máquina de estado, **602**
ámbito, VHDL, 526
Amerasekera, E. A., 921
amplificador, **150**
amplificador de sentido, 866
amplificador diferencial, **176**
amplificador separador, **80**
análisis
circuito combinacional, **193**
análisis de circuito combinacional, **193**, 209—214
análisis de temporización, 336, 759
herramientas, 337
analizador de temporización, 10
analógico, **3**
analógico *versus* digital, 3—6, 7—8, 79
ancho mínimo de pulso, **536**
anchura de pulso de escritura, **859**
anchura de pulso, mínimo, 536
AND, 93
AND alambrado, 132
AND, compuerta, 6, 82, 196, 204, 219
CMOS, 93
AND, operación, 197
AND-OR, circuito, 203, 212, 214, **218**, 219, 220, 223, 248, 346
AND-OR, dispositivo, 337
AND-OR-INVERT (AOI), compuerta, **94**, 185

AND-OR-INVERT (AOI), compuerta, CMOS, **94, 185**
AND, plano, PLA, **346**
anidados, IF-THEN-ELSE, ABEL, 630
anillo auxiliar de alimentación, **171**
ánodo, **146**
anotación posterior, 897
ANSI, *Véase* Instituto Nacional Americano de Estándares
apóstrofe o "primo" ('), **196**
arco dirigido, diagrama de estado, **556**
architecture, VHDL, 269
Archuleta, George, iv
argumentos, VHDL, **276**
aritmética de campos de Galois, 604
aritmética de complemento a dos, 39— 43
aritmética del complemento a uno, 44
arquitecto del sistema, 314
arquitectura, VHDL, **268**
arranque activo (pull-up), **127**
arranque pasivo, **127**
arranque, activo, **127**
arranque, pasivo, **127**
array, VHDL, 274
arreglo de compuertas, 17, 577, 614
arreglo de compuertas programables por campo, (FPGA, field-programmable gate array) 11, **15**, 16, 23, 313, **349**, 799, 882—891
arreglo de lógica programable (PLA, programmable logic array), **15**, **337**—340
  circuito bipolar, **346**
  diagrama, **338**
  salidas constantes, **339**
arreglo redundante de discos económicos (RAID), 67
arreglo, VHDL, **274**
  coincidencia de tipos, 286
ASCII *véase* Código Estándar Americano para el Intercambio de Información
ASIC, enrutado, 758
ASIC *véase* CI de aplicación específica
asignación de estado, 577, 581, 583
asignación de estado alterado, 571
asignación de estado codificado de salida, **551**, **598**, 798, 802, 812
asignación de estado lógico uno, **572**
  casi, 572
asignador del término del producto, **876**
asimetría de salida, TTL, 161
ASM *véase* máquina de estado algorítmico
ASM, diagrama, 645
asterisco, significado del, xvii
atenuador digital, 849, 849, 850
ATM, *véase* Modo Asincrónico de Transferencia
attributo, instrucción, VHDL, 816
atributo, sufijo de, ABEL, **391**, 628, 632
Auclair, Dan, 844
auto enrutador, 902
axioma, **195**

B

β, 151
BAJO (LOW), 3, 80, 86, **148**, 204, 317
banco de pruebas, 10, 266, 296, 300
bandera de conexión, 898
banderas de señal, 325, 327
Bartee. T. C., 298
base, 26
base, **26**
base binaria, 26
base, transistor, **149**
BCD, código, 351, 372
BCD, decodificador, **353**
BCD, suma, **50**
BCD, *véase* decimal codificado en binario
bebés infernales, 5
begin, VHDL, 269
Belgard, Richard A., 12
biblioteca, **280**
biblioteca de componentes, 897
biblioteca, VHDL, 280
biestable, **531**—534, 604
biestable J-K, 561, 666
bifásico, 72
Birkhoff, G., 298
Birkner, John, xxvi
bit, **26, 80**
bit, celda de, 69
bit de acarreo, 800
bit de datos, 332
bit de información, **59**
bit de mayor orden, 27
bit de orden bajo, **27**
bit de paridad, **59**
bit de signo, **34**, 37
bit de suma, 434
bit, línea de, **836**
bit más significativo (MSB, most significant bit), 27, 34, 37
bit menos significativo (LSB, least significant bit), **27**
bit, rapidez de, **69**
bit "sin importancia", **58**
bit, tiempo de, 69
bit, VHDL 271
bit_vector, VHDL, 271
bits de verificación, 60
bits por segundo, 69
BJT *véase* transistor bipolar de unión
Blakeslee, Thomas R. 784, 921
bloque de ecuación, ABEL, **254**
bloque de E/S (IOB, I/O block) **878**, **886**
bloque de función (FB, function block), Xilinx, 873
bloque lógico configurable (CLB, configurable logic block) 882
bloqueo de terminales, **881**
bloqueo de transistor Schottky, **154**
bloqueo inducido, **112**
bloqueo inducido, CMOS, **112**
Bolton, Martin, 785
BOM *véase* lista de materiales
Boole, George, 194, 298
boolean, VHDL, 271, 272
Booleana, álgebra, **194**, 298
  véase también álgebra de conmutación
borrado de memorias, 842
borrado de memorias EPROM, **842**
borrar, **535, 541**
BPRZ *véase* regreso a cero bipolar
bps, 69
Brayton. R. K. 300
broma, 530
  realmente mala, 602

Brown, Charlie ("Carlitos"), 534
búfer, separador (memoria de almacenamiento temporal/ memoria intermedia), 320
burbuja, 83, 90
burbuja de inversión, **83**, 90, 218, 315, **316**, 317, 319, 320—323, 356, 357
burbujas magnéticas, 855
bus, **315**, 326, 332, 898
  bidireccional, **391**
  de drenaje abierto, **131**
bus bidireccional, 391
bus bidireccional de datos, 861
bus de drenaje abierto, **131**
bus de tres estados, **126**, 670
  VHDL, 396
BUT, 305
BUT, compuerta, 305
BUT flop, **654**
byte, 28

C

C, 10, 11
C++, 10, 11
CA, carga de, **115**,758
CA, cargabilidad de salida, **110**
CAD *véase* diseño asistido por computadora
cadena de exploración, **544**
Cadence Design Systems, 265
CAE *véase* ingeniería asistida por computadora
caída de voltaje en el diodo, **147**
caídas del sistema Windows, xxv
cálculo del complemento de raíces, 36
cálculo del resistor de arranque, **133**
calificación de las partes, **897**
cámara digital, 3
cambios de entrada simultáneos, 604, 608, 609
Campbell, D. S., 921
campo, finito, 73, **730**
campos, diagramas esquemáticos, **897**
canal serial, **718**
Cantoni, Antonio, 785
capacidad de exploración, 543, 644
capacitancia de carga, 114, 171
capacitancia de carga, 171
capacitancia perdida, **114**
capacitancia, parásita, **114**
capacitor de consumo de potencia, **121**, 139
capacitores de desacoplamiento, **112**, 158
capacitores de filtrado, **112**
capacitores, desacoplamiento de, 112, 158
capacitores, filtrado de, 112
captura de diagramas esquemáticos, 896
característica de transferencia, 124
  compuerta de transmisión, **123**
característica de transferencia, CMOS, 99, 124
características de conmutación, TTL **168**
características eléctricas, TTL, 168
carburador, 4
caret ^, 197
carga
  CA, 115
  capacitiva, 115
  CD, 102, 115
  resistiva, **102**
carga, **141**
  cargabilidad total de salida, **161**
  circuito PLA, 346
  consumo de energía, 122
  etapa de salida, **157**
  margen de ruido, 160, 161
  salida de colector abierto, **165**
  salida de polo tótem, **157**
carga capacitiva, 114, 115, 333
carga CMOS, **141**
carga de máscara, **842**
carga de salida, 100, 101, 109, 110
carga eléctrica, 267
carga resistiva, 102
carga TTL, **141**
carga unitaria del estado ALTO, TTL, **161**
carga unitaria en estado BAJO (LOW), TTL, 160
cargabilidad de entrada, **92**
  TTL, 164
cargabilidad de salida, 96, 109, 127, 142, 170, 355, 758, 899
  CA, **110**
  CD, **110**
  estado ALTO (HIGH), **110**, 142
  estado BAJO (LOW), **109**, 142
  total, **110**
cargabilidad de salida del estado ALTO (HIGH), **110**, 142
cargabilidad de salida en estado BAJO (LOW), 109, 142
  TTL, 161
cargabilidad total de salida, **110**
  TTL, 161
carrera, **611**, 620, 750
carrera crítica, **611**, 611
carrera no crítica, **611**
CAS, latencia, 872
CAS-antes de-RAS, ciclo de actualización, **870**
CASE, ABEL, 630
Case, Brian, 12
CAS_L (column address strobe: selección de dirección de columna), **868**
cátodo, **146**, 146
causalidad, **330**, 535
cautín para soldar, 9
CCD, *véase* dispositivo acoplado por carga
CD, balance, 71, 733
CD, carga de, **102**, 115, 758
CD, cargabilidad de salida, **110**
CD, margen de ruido de, 96, **101**, 170
  estado ALTO (HIGH), 142
  estado BAJO (LOW), 142
  TTL, **160**
CD, margen de ruido de, TTL, 161
CD, *véase* disco compacto
celda 1 distinguida, 228, 235, 242
celda de prueba, **904**
células estándar, 312
$C_{ENTmáx}$, **140**
central telefónica particular (PBX, private branch exchange), 4
cerradura de combinación, 582
CI aplicación de pegamento, 14
CI de aplicación específica (ASIC, application-specific IC), **16**—17, 313, 577, 605, 750, 758
  diseño de, **14**, **23**, 312, 534

CI semipersonalizado, 16
CI *versus* chip, 12
circuito AND-OR de nivel dos, 214, **218**
circuito bipolar PLA, 346
circuito combinacional, **7, 82, 193, 529**
circuito de conferencia, 850
circuito de conferencia digital, 850
circuito de paridad impar, **413**
circuito de paridad par, **413**
circuito de restablecimiento, 565
circuito en modo de pulso, **645**
circuito en modo fundamental, 604, 606, 607
circuito equivalente de carga, **115**
circuito Hamlet, 309
circuito integrado (CI, integrated circuit), **12**—15, **84**
circuito iterativo, **421**, 424
  entrada en cascada, 747
  entradas de frontera, 747
  salida en cascada, 747
  salida primaria, 747
  salidas de frontera, 747
circuito lógico
  combinacional, **193**
  secuencial, **193**
circuito NAND-NAND de nivel dos, 214, **218**
circuito NOR-NOR, **219**, 220, 248
circuito NOR-NOR de nivel dos, **219**
circuito OR-AND de nivel dos, **219**
circuito para acarreo hacia delante, 441
circuito retenedor de bus, 670
circuito secuencial, **7, 82, 193, 529**
  no inicializado, 900
circuito secuencial de retroalimentación, 531, 534, 604—627
  riesgos de, 248
circuito secuencial no inicializado, **900**
circuitos iterativos, 747
círculo, 556
Cisco Systems, xxv
C_L, **122**
Clare, Christopher R., 645
clases de arreglo, VHDL, **274**
cláusula library, VHDL, 280
CLB *véase* bloque lógico configurable
Cluley, J. C., 921
CML *véase* lógica en modo de corriente
CMOS (MOS complementario), **85**—144, 176, 385, 758
CMOS de bajo voltaje (LVC, Low-Voltage CMOS), 174, 670
CO *véase* oficina central (central telefónica)
Coburn, James, 73
codificación de comprensión-expansión, **848**
codificación digital de voz, 852
codificador, **376**—378
codificador binario, **376**
codificador de prioridad, **377**, 47 1
código, **48**, 65, 351
código autocomplementado, **50**
código balanceado, 71
código binario de *n* bit, 351
código biquinario, 50
código de exceso 3, **50**
código de línea, 69
código de paridad impar, **60**
código de paridad par, **59**, 413
código de producto, **66**
código de verificación de suma, **68**
Código Estándar Americano para Intercambio de Información (American Standard Code for Information Interchange, ASCII), **53**
código Hamming, **61**, 415
código invertido 1 *fuera de n*, 55
código *m* fuera de *n*, 56
código para la corrección de error, 61, 415, 733
código para la detección de error, 58, 413, 733
código ponderado, 50
código reflejado, 52
códigos, 48—56
códigos decimales, 48—51
códigos para la detección de error, 58—68
coeficiente binomial, 49, 56, 240, 570
coeficiente de reflexión, ρ, 915
Cohen, Danny, 724
Colector, de transistor bipolar, **149**
com, palabra clave, ABEL, **250**
combinación de entrada sin importancia, 232, 236
  ABEL, 260
combinacional *versus* combinatorial, 343
combinatorial, 240
comentarios
  ABEL, **250**
  VHDL, **269**
comparación de números, 33
comparador, **419**, 476, 488
  iterativo, **421**
  paralelo, **421**
comparador de magnitud, **419**
comparador en paralelo, **421**
comparador iterativo, 421
comparador serial, **748**
comparador, serie, **748**
comparadores, 419—430
compensación (off-set), **209**, 260
compilador
compilador, 9
complemento, **196**
  de una expresión lógica, **202**
complemento a dos, **37**, 77
complemento a uno, **38**, 77
component, instrucción, VHDL, 282
component, VHDL, 283
componente de producto, **446, 448**
comportamiento de estado estacionario, **244**
comportamiento disparado por flanco, 540
comportamiento transitorio, **244**
compuerta, **6**
compuerta AND con diodo, **148, 156**
compuerta de inversión, 219, 355
compuerta de transición, 20, **123**, 462, 614
compuerta de un transistor CMOS, **87**
compuerta flotante, 349
compuerta habilitadora de salida, **342**
compuerta no inversora, 219, 355

compuerta NOR Exclusiva (XNOR), 410
comparador, 420
compuerta OR exclusiva (XOR), **304, 410**
comparador, 420
compuertas, 336
comunicación, 314
conceptos de electrónica, xvii
condición marginal de disparo, 534
condiciones de operación recomendadas, TTL, **168**
confiabilidad, 842, 908—912
confiabilidad del sistema, 921
configuración de emisor común, **150**
configuración de selección de chip, **859**
con habilitación, 542
conjunto completo, 305
conjunto completo de prueba, 903
conjunto d, 232, 260
conjunto de corte mínimo, **609**
conjunto de iniciación (on-set), **208,** 254, 260, 628
conjunto P (P-set), **300**
conjunto S, **300**
conjuntos rectangulares de 1, **224**
consenso, 200, **246**
consenso iterativo, 200
constant, VHDL, 272
constante de tiempo *RC*, **116,** 670
constante genérica, VHDL, **284**
constante, VHDL, 274
consumo
dinámico de energía, **121**, 139
en reposo de energía, **121**
estático de energía, **121**
consumo de potencia
CMOS, 96, 113
TTL, 122
consumo de potencia estática, **121**
consumo dinámico de potencia, 121, 139
consumo estático de energía, 121
contador, 693—707, 751
en ABEL, 705
contador arriba/abajo, **703**
contador binario, 705
contador binario de *n* bit, **694**
contador con autocorrección, **726**
contador de anillo, 704, 725, 736
con corrección automática, 726
contador de anillo con autocorrección 726, 900
contador de anillo trenzado, 727
contador de décadas, **699**
contador de división entre *m*, **693**
contador de funcionamiento libre, **698**
contador de Johnson, 727
autocorrección, 729
contador de Moebius, 727
contador de registro de corrimiento, 724
contador de rizo, 694
contador Johnson con autocorrección, **729**
contador LFSR, 730, 784
contador módulo *m*, **693**
contador paralelo sincrónico, **695**
contador serial sincrónico, **695**
contador sincrónico, **694**
control de polaridad de salida, 344, 486
controlabilidad, 907
controlador de salida simétrica, **137,** 336
controlador de semáforos, 53, 825
Véase también Sunnyvale
controlador de tres estados, **385**
convención de lógica negativa, **195,** 204
convención de lógica positiva, **195,** 204, 205, 317
convención de nomenclatura del nivel activo, **318**
conversión de base *r* a decimal, **29**
conversión de binario a hexadecimal, 27
conversión de binario a octal, 27
conversión de paralelo a serie, **712**
conversión de tipo explícito, VHDL, **510**
conversión de tipo, VHDL, **510**
conversión decimal a base r, **30**
conversión hexadecimal-binaria, **28**
conversión octal a binario, **28**
conversión serial a paralela, **712**
convertidor analógico-digital, 718
convertidor digital a analógico, 718
Conway, Lynn, 183, 645, 785
corrección de error, **61**
corriente
consumo de, 105
consumo TTL, **158**
de fuga, 88, 127, **147**
dirección, CMOS, 105
dirección, TTL, 160
flujo, CMOS, 105
flujo, TTL, 160
fuente de, 105
fuente TTL, **158**
corriente de consumo, 105
TTL, 158
corriente de fuente, **105**
TTL, 158
corriente de fuga, **88,** 101, 127, **147**
corrimiento aritmético, **743**
corrimiento circular, **743**
corrimiento tipo barril, **463, 468,** 500
corrimiento y resta, división con, **47**
corrimiento y suma, multiplicación con, 45
corrimiento y suma, multiplicador de, 753
corte (OFF), **151**
Cortner, J. Max, 921
Costello, D. J. Jr., 73
costo, 6, 16, 17, 22, 23, 409, 861, 902, 903, 908
circuito combinacional, 220, 225, 228
de diseños basados en PLD, 693
máquina de estado, 570, 572, 575, 579, 798, 802
costo de ingeniería no recurrente (NRE), **16,** 17, 23
CPLD, *véase* PLD complejo
CPU, *véase* unidad central de procesamiento
CRC, *véase* verificación de redundancia cíclica
cuadro, **720**
cubierta, **199**
cubiertas, **226**
cubo *n*, 57, **58**

cuerda
ABEL, 250
VHDL, 275
cuestiones térmicas, **901**
cuña, 196
curso, 243
cursos avanzados, xxi
cursos de introducción, xxi
cursos de laboratorio, xxi
CV^2f potencia, **122**, 136
character, VHDL, 271, 272
Charlie Brown ("Carlitos"), 534
Chicago, Illinois, 808, 825
chip, 6
chip versus CI, 12

D

D, latch, **538,** 666, 674, 854
dado, 12
plural de, 12
dados, 12
datos con separación doble, 723
datos en paralelo, **69**
datos seriales, **69**
Davidson, Edward S., xxvi
de, instrucción, ABEL, **260**
DDR-DRAM, DRAM sincrónica con doble unidad de transferencia de datos (double-data-rate synchronous DRAM), 891
decimal codificado en binario (BCD, binary-coded decimal), **48**
declaración constante, VHDL, **274**
declaración de componente, VHDL, 283
declaración de entidad, VHDL, **269**
declaración de función, VHDL, **281**
declaración de puerto, **270**
declaración de señal, VHDL, 271
declaración de variable, VHDL, **271**
declaración device, 250
declaración genérica, VHDL, **284**
declaraciones, ABEL, 249, **250**
Declaración de terminal, ABEL, **250**
decodificación, **61**
decodificación bidimensional, **839,** 860, 867
decodificador, **351**—374, 703, 833
decodificador binario, **352**, 405
decodificador de latch, 676
decodificador de siete segmentos, **372**
decodificador decimal, **353**
decodificador n a 2^n, 352
decodificador para la corrección de error, **64**
decodificar, **352**
definición de arquitectura, VHDL, **269,** 270
definición de función, VHDL, **276**
definiciones de terminal, ABEL, 363
DeMorgan, símbolos equivalentes de, 317
demultiplexor, **405**
depuración, 3, 111
derecha, dirección del registro de corrimiento, **716**
desafirmación, 317
desbordamiento, **43**
complemento a dos, **41,** 42
reglas, 41
descarga electrostática, 97
descarga electrostática, (ESD, electrostatic discharge), 112
descifrador, 733
descripción de comportamiento, VHDL, **289**
descripción de dispositivo lógico estructurado, **313**
descripción de flujo de datos, VHDL, **286**
descripción del circuito, 194, **313**
descripción estructural, VHDL, **283**
deshabilitador de compuerta, **510**
desplazador de nivel, 175
detección de fallas, 300
detector de números primos, 215, 221, 225, 232
Devo, 644, 785
DFT, *véase* diseño para verificación
diagonal (/), 315
diagrama de adyacencia, **620**
diagrama de adyacencia de estado, **620**
diagrama de bloques, 312, 313, 314, 324, 325, 405, 476
diagrama de estado, 553, 556, 560, 562, 564, 584—590, 591, 660
ABEL, 629—637
diagrama de estado ambiguo, **585**, 587, 590
diagrama de estado documentado mnemónico (MDS, mnemonic documented state), 645
diagrama de flujo, 645
diagrama de temporización, **83, 313, 330**, 558, 565, 567, 660, 662
diagrama de terminales, **13**
diagrama de Veitch, **298**
diagrama esquemático, **312**, 323—326
diagrama lógico, **312**, 323—326
dibujos provocativos, 558
digital, 3
digital *versus* analógico, 3—6, 7—8, 79
dígito binario, **25, 26, 80**
dígito de mayor orden, **26**
dígito de orden bajo, **26**
dígito más significativo, 26
dígito menos significativo, 26
dígitos hexadecimales A—F, **27**
Dillinger, Thomas E., 456
Diodo, **84**, **146**, 836
fijador de nivel, **173**
polarización directa, **146**
polarización inversa, **146**
Schottky, **154**
Zener, **147**
diodo con polarización directa, **146**
diodo con polarización inversa, **146**
diodo emisor de luz (LED, light-emitting diode), 102,129,130 161, 372
diodo fiador de nivel, **156, 173**
diodo Schottky, **154**
diodo semiconductor, 84, **145**
diodos fijadores de nivel, **918**
DIP, *véase* encapsulado de terminales en doble línea
disco compacto (CD, compact disc), 4, 81, 848
disco de codificación mecánica, 353
disco versátil digital (DVD, digital versatile disc), 4
diseñador de PCB, 901
diseñador lógico, 314
diseñadores digitales trabajando, xxii

diseño, 605
máquina de estado, 569, 577, 591
versus síntesis, **194**
diseño a nivel tarjeta del circuito, impreso, 22, 921
diseño al nivel de PCB, 22
diseño asincrónico, 2
diseño asistido por computadora (CAD computer-aided design), 9, 199, 317, 320, 895, 895
herramientas, 9—12, 895—902, 920
simulador analógico, 899
programa, 591
software, 590, 591, 758, 759
diseño basado en PLD, 312, 593, 600, 799
diseño de arreglo de compuertas, 17
diseño de célula estándar, 14, **17**
diseño de comportamiento, VHDL, **289**
diseño de flujo de datos, VHDL, **286**
diseño de latch de dos fases, **760**
diseño de lógica de burbuja, 2, **321**, 356, 359, 404, 412, 474
diseño de máquina de estado, 561, **563**, 569
diseño de máquina de estado *versus* programación, 564, 661
diseño de memoria finita, 801, 821
diseño de VLSI, 14
diseño digital, 1, 5
diseño estructural, VHDL, **283**
diseño jerárquico, 265
diseño lógico, 1, 5
diseño, máquina de estado, 561, **563**
diseño para las pruebas, (DFT, design for testability), 903
diseño para las pruebas, 902—908, 921
diseño sincrónico
riesgo del, 248
disparando el reloj, **760**
disparidad de corrida, 72
dispositivo acoplado por carga (CCD, charge-coupled device), 855
dispositivo de lógica programable (PLD, programmable logic device), 11, 14, 15, 16, 23, 313, 479-500, 549, 551, 563, 573, 577, 602, 681—692, 796-812
compilador, 391
complejo *véase* PLD complejo (CPLD, complex PLD)
programador, **349**
versus simulación, 11
dispositivo de lógica programable y borrable (EPLD), erasable programmable logic device), **348**
dispositivo de prueba, **904**
dispositivo iterativo, 747
dispositivo, serie, **747**
dispositivos GAL, 343—**344**, 685—693
dispositivos PAL bipolares, 687
distancia, **58**
distancia Hamming, **58**
distancia mínima, 59
distorsión cuantizada, 849
distribución de la multiplicación, 199, 211, 213, 216, 217, 432
división binaria, 47—48
división con signo, 48
división sin signo, **47**—48
divisor de fase, **156**, 164
doble de una expresión lógica, 203
documentación, 2, 3, 196, 312—330, 353—357, 536, 660—666
dos puntos (:), en el nombre del bus, 327
Downs, Thomas, 299
downto, VHDL, 273, 276
DPLL, *véase* lazo digital amarrado por fase
DRAM con salida extendida de datos, (EDO DRAW, extended-data-out), **871**
DRAM de modo de columna estática, **871**
DRAM de modo nibble, **871**
DRAM sincrónica (SDRAM), **871**
DRAM sincrónica con doble velocidad de transferencia de datos (DDR-DRAM, double-data-rate synchronous DRAM), 891
DRAM sincrónica mejorada (ESDRAM, enhanced synchronous DRAM), 891
DRAM, *véase* memoria RAM dinámica
drenaje, transistor CMOS, 87
DSL *véase línea* de suscriptor digital
dualidad, **195**, 203, 219
Duggal, Akhil, xxvi
DVD, *véase* disco versátil digital

E

easy-ware herramienta fácil de usar, 896
eclipse, **228, 243**
ECL positivo (PECL, positive ECL), **182**
ecuación característica, 548, **552**, 558, 559, 562, 573, 577, 593
ecuación de excitación, **554**, 559, 561, 593, 605
ecuación de polarización inversa, 252
ecuación de salida, **555**, 560, 562, **607**
ecuación de transición, 555, 559, 562, 591, 593
ecuación intermedia, ABEL, 252, 483
ecuación lógica, **318**
ecuaciones, ABEL, **250**
polaridad inversa, 252
ecuaciones, ABEL, variable de estado en el lado izquierdo, 630
ecuaciones de costo mínimo, 572, 575, 578, 579
ecuaciones de riesgo mínimo, 572, 575, 578
editor de diagramas esquemáticos, 896
editor de texto, VHDL, **265**
EDO-DRAM, *véase* DRAM con salida extendida de datos
Edsall, Tom, xxvi
EEPROM, *véase* memoria sólo de lectura eléctricamente borrable y programable
efecto de campo, 87
ejemplificar, en forma concreta, 268, **282**
electrónica analógica, 1

elemento lógico de gran escala, 319, 323, 353—354
elemento OR fijo (FOE, fixed-OR element), **340**
elementos sin importancia en la codificación de estado, 600
elementos sin importancia en la notación de tablas de verdad, 352
elementos sin importancia en las tablas de excitación, 623
ELSE, instrucción, ABEL, 253, 630
else, VHDL, 269, 287, 292
elsif, VHDL, **292**
el tema divertido, **xvii,** 2, 577, 604, 896
emisiones de radio frecuencia, **901**
emisor, de transistor bipolar, **149**
empaquetador, 268
emprendedor, 314
ENABLE, palabra clave, ABEL, 252
encapsulado de terminales en doble línea (DIP, dual in-line-pin package), 13, 330, 667
encapsulado tipo DIP, 353
end, instrucción, ABEL, **252**
end, VHDL, 269, **270**
enfriamiento, 176
Eniac, 84
enlace T1, 72
enrutado de PCB, 758
enrutamiento, 902
entidad, VHDL, **268**
entity, VHDL, 269, **270**
entrada, **854**
entrada
  con tolerancia a 5 V, 173, 174
  flotante, **111**
  PLA, **337**
entrada asimétrica, **178**
entrada asincrónica, 750
entrada de borrado, 660, 905
entrada de comando, **751**
entrada de condición, 751
entrada de corte de energía, **846**
entrada de dirección, **832**
entrada de disparador Schmitt, **124**, 765
entrada de prueba, TI (test input), 544
entrada de pulso, **645**
entrada de retroalimentación, **690**
entrada de selección de chip (CS, chip-select), **845**
entrada deshabilitadora de salida, 164
entrada en cascada, circuito iterativo, **747**
entrada esquemática, 9
entrada flotante, **111**, 670
  TTL, 163
entrada habilitadora, 352, 357
  decodificador 359
  flip flop D, 542
  multiplexor, 402, 405
  separador de tres estados, 389
entrada habilitadora de reloj, **542**
entrada habilitadora de salida (OE, output-enable), 164, **845**
entrada serial, registro de corrimiento, **712**
entradas asincrónicas flip-flop, **541**
entradas de dirección multiplexadas, **868**
entradas de frontera, circuito iterativo, **421,** 747
entradas diferenciales, **177**
entradas en cascada, comparador, 423
entradas y salidas en cascada, circuito iterativo, **421**
entradas y salidas primarias, circuito iterativo, **421**
EPLD), **348**
EPROM tipo flash, 843, 844
EPROM *véase* memoria sólo de lectura borrable y programable,
equations, instrucción, ABEL, **250**
Equipo automático de prueba 904
Equivalence, compuerta, **410**
equivalente de Thévenin, **103**
error, **58**
error de desbordamiento (overflow) en la división, **48**
error múltiple, **58**
error simple, **58**
error unidireccional, **68**
errores, 5, 272, 326
errores bobos, 899, **902**
errores en este libro, xxv
escenario, **901**
escritura, 314
escritura controlada por CS, 859
escritura controlada por WE, **859**
ESD *véase* descarga electrostática
ESDRAM (DRAM síncrona mejorada), 891
Eshraghian, Kamran, 183
eslabón fusible, **346, 842**
especificación del circuito, **312**
especificación máxima absoluta, 173
especificación máxima absoluta, TTL, **168**
especilficaciones, **8**, 100, 312
especificaciones de temporización, 660
Espresso-EXACT, 300
Espresso-II, 244, 300
Espresso-MV, 244, 300
Esquemático, diagrama, 283
  jerárquico, 898
estable, **533**
establecer, 535
estación de trabajo de ingeniería, **897**
estado, 7, **530**, 553
  anormal, 600, 725, 726
  codificado, **569**
  inactivo, 569
  inicial, 571, 572
  no usado, 600, 632
  seguro, 632, 726
estado anormal, **600**, 725, 726
estado codificado, **569**, 587, 620
estado de alta impedancia, **126**, 385
estado de alta impedancia (Hi-Z), **126**, 385, 386, 387
estado de entrada, **607**
estado de espera, 569
estado flotante, **126**, 385
estado inicial, **566**, 570, 571, 572, 900
estado interno, **607**
estado nombrado, 587, 620
estado seguro, 598, 632, 726
estado total, **607**, 608
estado total estable, **607**, 618
estado total inestable, **607**
estados adyacentes, **620**
estados codificados, 620
estados compatibles, 614
estados equivalentes, **568**
estados, no usados, **569**, 571, **572**
estados, número total de, **569**
estados sin importancia, 598

estados sin usar, **569**, 571, 572, 600, 632
estructura esquemática jerárquica, **325**, 898
estructura esquemática plana, **325**
etapa de salida, TTL, **157**
Ethernet, 733
  gigabit, 176
event, atributo, VHDL, **643, 678**
excitación, **554**
excitación de salida asimétrica, **137**
exclusión mutua, **561**, **587**
exit, instrucción, VHDL, **293**
exponente, 848
expresión
  álgebra binaria, 196, 197
  producto de sumas, 203, 207, 214, 219
  producto de sumas mínimo, 231
  relacional de ABEL, 258
  suma de productos, 203, 207, 211, 214, 216, 217, 219, 299
  suma de productos, mínimo, 231
expresión ambigua, 198
expresión constante, ABEL, 362
expresión de producto de sumas, 203, **207**, 212, 214, 219
expresión de signo, 35, 37, 47, 76
expresión de transición, 560, 561
expresión del mínimo producto de sumas, 231
expresión lógica, 197, **318**
  complemento de, **202**
  doble de, **203**
  entre paréntesis, 211
expresión lógica entre paréntesis, 211
expresión mínima de la suma de productos, 231
expresión relacional, ABEL, 258
expresión, ambigua, 198
extensión de signo, 35, 37, 47, 76

F

FALSE, 3
false, VHDL, **272**
falla, **58**
  intermitente, 112
falla intermitente, 112
falla permanente, **58**
falla persistente simple, **903**
falla temporal, **58**
familia 100K, **182**
familia 10H, **179**
familia 10K, **179**
familia lógica, **85**
familia lógica bipolar, **85**
familias lógicas pequeñas, 669
Farley, Rebecca A., xxvi
FCC Parte 15, **901**
FCT (CMOS rápido, compatible con TTL), 142
FCT-T (CMOS rápido TTL, compatible con TTL V_{OH}), 143
Feinsmith, Jason, xxvi
FIFO, *véase* memoria primeras entradas, primeras salidas (first-in, first-out)
filósofos griegos, 644, 785
Fiorino, Mike, iv
FIT, **909**
flecha. Diagrama de estado, 556
Fleischer, Bruce M., xxiv, 183
Fletcher, William I., 645
flip-flop de exploración, 907
flip-flop, 7, (**534**, 540-549, 604, 666
  entradas asincrónicas, **541**
  CMOS D, 614
flip-flop (dispositivo biestable) T, **549**, 694, 696
  con habilitación, 549
flip-flop D disparado por flanco, 531, 540-545, 677
flip-flop D disparado por flanco negativo **541**
Flip-flop D disparado por flanco positivo, **540**—545
flip-flop disparado por pulso, **545**
flip-flop J-K disparado por flanco, **547**
flip-flop J-K maestro/esclavo, **546**
flip-flop S-R maestro/esclavo, **545**
flip-flop-D, **540—545**
  CMOS, 614
  con habilitación, **542**
flip-flops enterrados, 602
flujo de diseño, **264**
  VHDL, 264
Flynn, Michael J., 73
$f_{máx}$, 691
for, lazo de iterativo, VHDL, **293**
Ford Thunderbird, 585
fórmula de expansión anidada, 30
Foundation express, herramientas de diseño digital de, xxii
FPGA *véase* arreglo de compuertas programables por campo
Franaszek, Peter, **74**
frecuencia de falla en los componentes, 909, 911
f, frecuencia de sincronizador, **768**
frecuencia de transición, **121**, 139
fuente de alimentación, 112
fuente de transistor CMOS, 87
función AND con diodo, 346
función de estado siguiente, 553
función de mayoría, **454**
función de producto m, **234**
función de resolución, 396
función de resolución, VHDL, 396
función de salida, máquina de estado, 553
función de salida múltiple, 236
función de transferencia, 532
función impura, **526**
  bus de tres estados, 396
  carrera, 290
  ciclo de simulación, **296**
  conversión de tipo, **510**
  cuerda, **275**
  declaración de señal, 271
  declaración de variable, 271
  descripción estructural, **283**
  diseño estructural, **283**
  editor de textos, **265**
  encapsulado, 280
  latch inferido, **677**
    función de resolución, 396
    función pura, **526**
  herramientas de síntesis, **264**
  in, instrucción, 269, **270**
  inout, instrucción **270**
  instrucción, **269**
  instrucción de asignación de señal secuencial, **290**
  instrucción de asignación de señal seleccionada, **287**
  instrucción de asignación de variable, 290

integer, tipo, 271, **272**
intervalo, 273
is, instrucción, 269, 270
library, cláusula, **280**
lista de sensibilidades, **290**
natural, subtipo, **274**
next, instrucción, **293**
not, operador, 269
null, instrucción, **278**
operador desigualdad (/=), **427**
operadores relaciónales, 287
others, instrucción, **275**, **288**, 292, 396
out, instrucción, 269, **270**
package body, instrucciones, **281**
package, instrucción, **281**
palabras reservadas, 269
port, instrucción 269, 270
port map, instrucciones, 282, **283**
precedencia de operador, 287
procedimiento, **279**
proceso, **289**
proceso, instrucción, **289**
process, instrucción, **289**
puerto, **270**
range, atributo, **279**
range, instrucción, 273
resultado, **276**
return, instrucción, **276**, **277**
rol, 502
ror, 502
salida de tres estados, 395
select, instrucción, 367
señal versus variable, **747**
signal, instrucción, 271
simulador, **266**
sla, 502
sll, 502
sobrecarga de operador, **277**, 287
sra, 502
srl, 502
std_logic, 396
std_logic, tipo, **273**
std_logic_1164, 281
std_logic_Arit., 281
std_logic_signed, 281
std_logic_unsigned, 281
std_logic_vector, tipo, **276**
std_ulogic, 396
subtipo, **273**, **396**
subtype, instrucción, **272**
suspendido, **290**
tiempo de simulación, **296**
tipeado fuerte, **272**
tipo arreglo no restringido, **276**
tipo definido por el usuario, 272
tipo no resuelto, **396**
tipos predefinidos, **271**
to, 273, 276
true, 272
type, **271**
type, instrucción, **272**
use, cláusula, **280**
variable, **271**, **289**
variable, instrucción, 271, **290**
variable *versus* señal, 747
wait, instrucción, **295**
when, instrucción, 269, 287, **292**
while, ciclo, **295**
work, librería, **280**, 522
función impura, VHDL, **526**
función lógica autoduplicada, 305
función OR Exclusiva, 231
función pura, VHDL, 526
función techo, **53**
función umbral, **454**
función, VHDL, **276**
impura, **526**
pura, **526**
funciones de producto, 234
funciones estándar, 14
funciones estándar MSI, 312
function, VHDL, **276**
fusible de seguridad, 351
fusibles de control de arquitectura, GAL16V8, 344

G

Gagliardi, R. M., 74
GAL16V8, **344**, 486, 690, 692
GAL16V8C, **344**, **685**, 687
GAL16V8R, **685**
GAL16V8S, **685**, 687
GAL20V8, **344**, **687**, 692
GAL22V10, **689**, 692
Galois, campos, 784
Galois, Évariste, 73,730
ganancia, **151**
Gateway Design Automation, 265
Gaubatz, Donald A., xxii
generación de acarreo, 434
generación de entrada de prueba, 733
generador de secuencia de longitud máxima, 730
generador de secuencia pseudoaleatoria, 603, 604
generador de temporización, 736
generate, instrucción, VHDL, **283**, **453**
generic map, VHDL, **284**
generic, VHDL, **284**
G (giga-), 72
gigabit Ethernet, 176
Gill, John, xxvi
Goldstine, Herman H., 298
gongs, 455
GOTO, instrucción, ABEL, **630**
Grabel, A., 183, 456
Graham, Martín, 921
Gray, código, **51**, 57, 352
Gwennap, Linley, xxvi, 12

H

habilitación de escritura (WE, write-enable), **854**
habilitación de prueba (TE, test enable), **544**, 907
habilitación de tres estados, **385**
Hachtel, G. D., 300
Hamming, R. W., 61
hardware reconfigurable, 349
Hayes, John P., 921
HC (CMOS de alta velocidad), **136**, 173
HCT (CMOS de alta velocidad, compatible con TTL), **136**, 669
Heater, Courtenay, xxii
Hellerman, Herbert, 530
herramientas de síntesis, VHDL, **264**
herramientas de software, 9—12, 799
para diseño lógico, 196
herramientas del diseño asistido por computadora (CAD), 895—902
hertz, 69
HIGH 3, **80**, 86, **148**, 204, 317
histéresis, **124**, 328, 389
HM62256, **861**
HM6264, **861**

HM628128, **861**
HM628512, **861**
hoja de datos, **97**
Huntington, E. V., 298
Hwang, Kai, 73
Hz, 69

I

I_{CCL}, TTL, **168**
I_{CCT}, 139
I_{CCH}, TTL, **168**
Identificador
  ABEL, **249**
  VHDL, **269**
IEEE 1076, paquetes aritméticos, 281
IEEE 1164, paquete lógico estándar, 273, 276, 277, 281, 395
IEEE Instituto de Ingenieros Eléctricos y Electrónicos (Institute of Electrical and Electronic Engineers), **316**
IEEE símbolos lógicos estándar, 455, 661
IF, instrucción, ABEL, **630**
if, instrucción, VHDL, **291**
I_I, TTL, **168**
I_{IH}, **101**
I_{IL}, **101**
$I_{Imáx}$ **140**
imagen grande, 23
impedancia característica, 192
impedancia característica Z_0, 913
impedancia de terminación, **915**
impedancia *versus* resistencia, 88
implementación con compuertas BUT, **459**
implicación, **225**
implicante primo, **226**
  de salida múltiple, **234**
  esencial, **228**, **235**, 242
  esencial secundario, **229**, 243
  redundante, **243**
implicante primo de salida múltiple, **234**
implicante primo esencial, **228**, **235**, 242
implicante primo esencial secundario, **229**, 243
implicante primo redundante, **243**
impuestos, 809, 827
in, VHDL, 269, **270**
inclusión, **226**
inclusión total, 561, 587
indicador de entrada dinámica, **540**
indicador de referencia, **328**, **897**
indicador de salida pospuesta, **545**, 660
indicador dinámico, 660
índice, 24
índice de arreglo, VHDL, **274**
inducción finita, **200**
inducción perfecta, **198**, 206
ingeniería asistida por computadora, (CAE, computer-aided engineering), 895
ingeniería no recurrente, (NRE)
  costo, 16, 17, 23
inhibición del rebote, **668**
inmunidad al ruido, ECL, 177
inout, VHDL, **270**
Instituto Nacional Americano de Estándares (American National Standards Institute, ANSI), 316, 455
instrucción case, VHDL, **292**
instrucción concurrente, VHDL, 282
instrucción de asignación de señal concurrente, VHDL, 286
instrucción de asignación de señal concurrente condicional, VHDL, 287
instrucción de asignación de señal secuencial, VHDL, **290**
instrucción de asignación de señal seleccionada, VHDL, 287
instrucción de asignación de variable, VHDL, **290**
instrucción de transición, **630**
instructores, xxi
integer, VHDL, 271, **272**
integración a escala diminuta, 14
integración a gran escala, (LSI), 14, 15
  funciones, 17
integración a mediana escala (MSI, medium-scale integration), **14**, 23, 362 funciones, 14, 17, 312, 467—477
integración a muy grande escala, (VLSI), **15**, 243
  diseño, 14
  personalizada, 348
integración a pequeña escala (SSI, small-scale integration), 13, 23, 362, 577, 666
intercambio espacio/temporal, 718, 747, 749, 784
  interfaz entre CMOS/TTL, 102, 105, 136
interferencia ("glitch"), 2, **244**, 340, 703, 727, 729, 761, 797
interferencias en la decodificación, **703**, 727, 729
interruptor, 667
interruptor tipo DIP, **667**
intervalo, ABEL, 258
intervalo dinámico, **848**
intervalo, VHDL, 273
inversión de marca alternativa (AMI, Alternate Mark Inversion), 72
inversor, **7**, **82**, 88—90 152—154, 196, 316, 320
  CMOS, 88
inversor de disparador Schmitt, 837
inversor lógico, 152
  CMOS, 88
inversor, CMOS, 88—90
invert, palabra clave, ABEL, 252
I_{OH}, 101
$I_{OHmáx}$, 104
$I_{OHmáxC}$, 142
$I_{OHmáx}$, TTL, **161**
I_{OL}, 101
$I_{OLmáx}$, 104
$I_{OLmáxC}$, 141
$I_{OLmáxT}$, TTL, **161**
I_{OS}, TTL, **168**
IP (Protocol Internet), 4
is, VHDL, 269, **270**
ISDN (Integrated Services Digital
istype, palabra clave, ABEL, 250
izquierda, dirección de registro de corrimiento, 716

J

Jain, Prem, xxvi
Jenkins, Jesse, xxvi

Johnson, Howard W., 921
JPEG, 4
juego de adivinanzas, 594, 602—603, 804—808, 823—825
juego de palabras al margen, xxiii

K

K (kilo-), 72
Kane, Patrick, xxvi
Karnaugh, M., 299
kilo-(K), 72
Kleeman, Lindsay, 785
Klir, George J., 183, 299
Knuth, Donald E., v, 73, 343
Kohavi, Zvi, 300, 645

L

λ, 909
Lancaster, Don, 183
Lansen, Ib, 183, 784
latch, 534, 535—540, 604, 666
  inferido de VHDL, 677
latch de renglón, 869
latch esclavo, **540**
latch maestro, **540**
latch S-R, **535**, 674
  con habilitación, **538**
latch (S-R) con barra, 537
latch transparente, **539**, 854
lazo de actualización, **867**
lazo de actualización de sólo RAS, 869
lazo de escritura, **856**, **870**
lazo de escritura en modo de página, **871**
lazo de lectura en modo de página, **870**
lazo de simulación, VHDL, 296, 297
lazo de trabajo, **530**, 699, 701
lazo lectura-modificación-escritura, 870
lazo de retroalimentación, 193, 531, 532, 533, 534, 535, 537, 604, 605, 609, 615
lazo digital amarrado por fase, 774
lazo digital amarrado por fase (DPLL, digital phase-locked loop) **71**, 774
LCD pantalla de cristal líquido (liquid-crystal display), 372

pág. 937

mapeo unívoco (de uno a uno), **351**
máquina de estado, 550—584, 750, 764
  costo, 572, 575
  diseño, 577, 591
  restablecimiento, 569
  síntesis, 591
máquina de estado algorítmico (ASM, algorithmic state machine), **645**
  gráfica, 645
máquina de estado alterado, 755
máquina de estado con latch de combinación, 800, 820
máquina de estado, descompuesta, 755
  descomposición de máquina de estado, **602**
máquina de estado finito, **530**, 568, 653
máquina de estado sincrónica temporizada, **531**, **550**
máquina de Mealy, 551, 555, 556, 557, 566, 568, 582, 605

metateorema, **203**
método de exploración, 907, 921
método de ramificación, **229**, **243**
método de trayectoria de exploración, 907
metodología del diseño sincrónico, 663
metralla, 347

pág. 938

modelo, hardware, 5
módem, 733
modo activo, **846**
Modo Asincrónico de Transferencia (ATM, Asynchronous Transfer Mode), 176
Modo de alerta, **846**

Pág. 940

RAM dinámica (DRAM), 854, 866—872
  sincrónica (SDRAM), 871
  temporización astuta, **869**
RAM estática (SRAM), **854**—861
  asincrónica, **861**
  célula, **856**

comandado, 714
distribución, 177
en sistema sincrónico, 750
fluctuación, inestabilidad de, 664
frecuencia, **530**
marca de, **530**, 550
periodo, **530**

pág. 941

dinámico, **247**, 750
en circuito secuencial de retroalimentación, 248
en diseño sincrónico, 248
estático, 674, 703, 750
estático-0, **245**
estático-1, **244**
función, **704**, 750, 784
fundamental, **624**, 677, 757

  de colector abierto, 127, 328
  de drenaje abierto, **127**, 129, 130
  tolerante a 5 V, 174
salida auxiliar, PLD, **364**
salida de acarreo combinacional, 707
salida de acarreo con registros, 707
salida de colector abierto, 127, **165**, 328
salida de datos, **832**
salida de drenaje abierto, **127**, 129, 130, 328
salida de multiplexor, 402
salida de totem pole, TTL, **157**
salida de tres estados, **126**, 164, 402, 670
  VHDL, 395
salida en cascada, circuito iterativo, **747**

M

M (mega-), 72
macrocelda de lógica de salida, **687**
macrocelda enterrada, **873**
macros, 312
Manchester, código, 72
Mano, M. Morris, 298, 645
Mazzola, Mario, xxv
mantisa, 848
mapa de Karnaugh, **221**
  de 5 variables, 307, 574
  de 6 variables, 308
mapas de excitación, **574**
mapeo unívoco (de uno a uno), **351**
máquina de estado, 550—584, 750, 764
  costo, 572, 575
  diseño, 577, 591
  restablecimiento, 569
  síntesis, 591
máquina de estado algorítmico (ASM, algorithmic state machine), **645**
  gráfica, 645
máquina de estado alterado, 755
máquina de estado con latch de combinación, 800, 820
máquina de estado, descompuesta, 755
  descomposición de máquina de estado, **602**
máquina de estado finito, **530**, 568, 653
máquina de estado sincrónica temporizada, **531**, **550**
máquina de Mealy, 551, 555, 556, 557, 566, 568, 582, 605
máquina de memoria finita, **653**
máquina de Moore, 551, 556, 557, 560, 565, 587 605
máquina de Turing, 530
máquina principal, **602**
marcas registradas, 340
margen de ruido, **8**, 154, 170
  CD, 96, 101
  TTL, 160, 161
margen de ruido de CD del estado ALTO (HIGH), 142
margen de ruido de CD en estado BAJO (LOW), 142
margen de temporización, **663**
margen de tiempo de configuración, **663**
margen para el tiempo de retención, **663**
márgenes de diseño de la ingeniería, **96**
márgenes, diseño de ingeniería, **96**
Marquand, A., 298
Marte, 76
máscara, **841**
material tipo *n*, **145**
material tipo *p*, **145**
Matick, Richard, **74**
matiz de mercadotecnia, 5—6
matriz de conmutación programable (PSM, programmable switch matrix), **889**
matriz de verificación de paridad **61**
maxitérmino, 207, 220, 230
maxitérmino *i*, 208
McClure, James, xxvi
McCluskey, Edward J., xxvi, 236, 298, 299, 300, 644, 645,
McFadden, Robert, xxvi
MCM (multichip module, módulo de chip múltiple), **18**
McMullen, C., 300
Mead, Carver, 183, **645**, 785
mecanismo de desgaste, **910**
media suma, **431**
medio sensible a la transición, **71**
M (mega) 72
memoria, 7, 14, 414
  primeras entradas, primeras salidas (FIFO, first-in, first-out), 891
memoria de acceso aleatorio (RAM random-access memory) **854**-872, 891
  estática (SRAM) **854**-861
memoria de acceso serial, **855**
memoria de estado, **550**
memoria de lectura/escritura (RAM), 676, 751
memoria de lectura/escritura (RWM), 854
memoria flash, 844
memoria no volátil, **833**, 842, **854**
memoria primeras entradas, primeras salidas (FIFO, first-in, first-out), **891**
memoria programable sólo de lectura (PROM, programmable read-only memory), **349**, **842**
  bipolar, 842
  programador, **349**, **842**
memoria sólo de lectura (ROM), 676, **832**—853
  programable una vez (OTP), 843
memoria sólo de lectura eléctricamente borrable y programmable, EEPROM, **843**
memoria sólo de lectura programable y borrable (EPROM, erasable programmable read-only memory), **842**, 910
memoria volátil, 854
Mercury Capri, 585
metaestabilidad, 536, 537, 538. 540, 541, 546, 609, 644, 757, 762, 764—773, 891
metaestable, 533, **764**
metateorema, **203**
método de exploración, 907, 921
método de ramificación, **229**, **243**
método de trayectoria de exploración, 907
metodología del diseño sincrónico, 663
metralla, 347
microprocesador, 14, 326, 376, 389, 414, 531, 768, 772, 831, 843, 844, 845, 909
Microprocessor Report, 12
microsegundo (μsec), 6
Michels, Diana, 12
Michelson, A. M., 74, 784
milésima de pulgada (mils), 18
miles y miles de millones, 48, 55, 240, 311, 456, 465, 853, 893
Millman, J., 183, 456
minicomputdora PDP-11, 837
minimización, 220
minimización de estado, 614, 619—620
minimización de salida múltiple, 233—235
minitérmino, 207, 220, 221, 223
minitérmino *i*, 208

minuendo, 32
modelo de componente, 899
modelo de comportamiento, 900
modelo de diseño lógico, 9
modelo de error, **58**
modelo de error independiente, **58**
modelo de hardware, **5**, **900**
modelo de interruptor, CMOS, 89
modelo de software, 900
modelo estructural, **900**
modelo físico, **900**
modelo, hardware, 5
módem, 733
modo activo, **846**
Modo Asincrónico de Transferencia (ATM, Asynchronous Transfer Mode), 176
modo de alerta, **846**
modo de columna estática, **871**
modo de exploración, 907
modo nibble, **871**
modo ráfaga, 863
module, instrucción, ABEL, **249**
módulo, **693**
módulo de chips múltiples (MCM), 18
módulo ROM de carga de sistema, 837
montador, **267**, **799**
mortalidad infantil, **910**
MOS complementario (CMOS, complementary MOS)
  carga, **141**
    interfaz con TTL, 102, 105, 136
MOS complementario *véase* CMOS
MOS, ROM, **840**
MOS, transistor, 85
  compuerta flotante, 348
MOSFET (transistor de efecto de campo con semiconductor de óxido y metal), 85—88
Mott, T. H., Jr., 299
MPEG-2, 4
Mr. Bean, 534
MSB, *véase* bit más significativo
MSI *véase* integración a mediana escala
MTBF *véase* tiempo medio entre fallas
muestreo digital de voz, 848
multiplex, **718**
multiplexor, 19, **398**—409, 473
  bloque de construcción MSI, 20
  CMOS, 19
  diseño a nivel compuerta, 20
  ecuación Booleana, 20
  entrada de habilitación, 402, 405
  modelo de interruptor, 19
  programa ABEL, 21
  programa VHDL, 21
  realización con PLD, 21
  tabla de verdad, 20
multiplexor con selección de salida, 633
multiplicación binaria sin signo, **45**
multiplicación con signo, 46—47
multiplicación de complemento a dos, **46**
multiplicación lógica, **196**
multiplicación sin signo, 45—46
multiplicador binario de velocidad, **789**
multiplicador combinacional, **446**
multiplicador, corrimiento y suma, **753**
multiplicador secuencial, **448**
multiplicadores, 446—453
Muroga, Saburo, 183
mux, **398**

N

NAND, compuerta, **83**, 90, 201, 219, 316, 537
  CMOS, 90
NAND-NAND, circuito, 214, 218, 219, 220, 248, 346
nanosegundo (ns), **6**
natural, VHDL, **274**
NBUT, compuerta, **654**
neg, palabra clave, ABEL, 252
negar, **317**
nerds, 195, 301
Network, (Red Digital de Servicios Integrados), 718
next, instrucción, VHDL **293**
nibble (dígito hexadecimal de 4 bits), **29**
nivel activo, 2, **317**, 321, 323, 354, 356, 473
nivel lógico inválido, 8
nivel lógico no definido, 8
nivel lógico no válido, 8
NMOS (MOS de canal-*n*), **87**
nodo, diagrama de estado, **556**
nombre de estado, **555**, 556
nombre de señal, 318, 323, 354, 357, **898**
NOR, compuerta, 83, 90, 202, 219, 316, 537
  CMOS, 90
No retorno a cero con inversión en 1 (NRZI, Non-Return-to-Zero Invert-on-1), **71**
No retorno a cero (NRZ, Non-Return-to-Zero), **69**
NOT, compuerta, **7**, **82**
NOT, operación, **196**
NOT, prefijo, **250**
not, VHDL, 269
notación, 138
notación con apóstrofo, 196
notación con barra sobrepuesta, 196, 356, 357
notación con tilde, 196
notación de tabla de verdad, 356, 401
notas al margen, xxii
npn, transistor, **150**
NRE *véase* costo de ingeniería no recurrente
NRZ *véase* no retorno a cero
NRZI *véase* no retorno a cero con inversión en 1
null, instrucción, VHDL, **278**
número de minitérmino, 208, 221, 236
número de parte aprobada, **897**
número de terminales, **328**, **897**
número extra negativo, 37, 42
número, total de estados, **569**
números BCD negativos, 49,
números con signo versus números sin signo, **43**
números negativos, 34—39

O

oblea, **12**
observabilidad, **904**, 907
octal, **387**
oficina central (CO, central office), **4**, **718**

operador algebraico, **196**
operador binario, **199**
operador de asignación temporizado, :=, **628**
operador de concatenación &, VHDL, 276, 502, 743
operador de tabla de verdad no temporizado, —>, 257
operador de tabla de verdad temporizado, :>, **628**
operadores de relación
  ABEL, 258
  VHDL, 287
OR plano, PLA, **346**
OR, compuerta, **7**, **82**, 93 197, 204, 219
  CMOS, 93
OR, operación, **197**
OR-AND, circuito, 203, 212, **219**, 220, 248
OR-AND-INVERT (OAI), compuerta, **95**, 185
OR-AND-INVERT (OAI), compuerta, CMOS, **95**, 185
Osborne, Thomas E., 645
oscilación amortiguada, 905, **913**
others, VHDL, **275**, **288**, 292, 396
OTP-ROM (ROM programable una vez one-time programmable ROM), **843**
otras instrucciones, ABEL, **250**
out, VHDL, 269, **270**

P

package body, VHDL, **281**
package, VHDL, **281**
PAL10H8, 687
PAL10L8, 687
PAL12H6, 687
PAL12L6, 687
PAL14H4, 687
PAL14L4, 687
PAL16H2, 687
PAL16L2, 687
PAL16L8, **340**, 342—343, 344, 362, 364, 405, 486, 674, 681, **682**, 692
PAL16R4, **682**
PAL16R6, **682**
PAL16R8, **681**, **682**
PAL16Rx, 692
PAL20L8, **343**, 344, 483, **682**, 692
PAL20R4, **682**
PAL20R6, **685**
PAL20R8, **685**
PAL20Rx, 692
palabra clave, VHDL, 269
palabra de código **48**
palabra sin código, **58**
palabras reservadas, VHDL, 269
PALCE16V8, **344**, 690, 692
PALCE20V8, **344**, 692
PALCE22V10, 692
pantalla de cristal líquido (LCD, liquid-crystal display), 372
pantalla de siete segmentos, 372
paquete lógico estándar, IEEE 1164, 273, 276, 277, 281
paquete, VHDL, **280**
paquetes aritméticos, IEEE 1076, 281
parámetros formales, **276**
  constante genérica, **284**
  declaración de función, **281**
  declaración genérica, **284**
  definición de función, **276**
  función, **276**
  function, instrucción, **276**
  generate, instrucción, **283**, **453**
  generic, instrucción, **284**
  generic map, cláusula, **284**
  identificadores, **269**
  IEEE 1076, paquetes aritméticos, 281
  IEEE 1164, paquete lógico estándar, 273, 276, 277, 281
  if, instrucción, **291**
parámetros formales, VHDL, 276
parámetros reales, VHDL, **276**
Parker, Kenneth P., 921
particionador, **799**
paso base, **200**
paso de inducción, **200**
pasta de soldadura, 18
patentes, 74
PCB *véase* tarjeta de circuito impreso
películas, 5
Pellerin, David, 265
penalización de retorno, SSRAM, **864**
pepperoni, 12
perfume, 100
PERL, 10, 11
peso, **26**, 37, 46
  del MSB, 37
Peterson, W. W., 73
pico o punta, **904**
picos de corriente, **112**
picosegundo (ps), **6**
pista, PCB, 18
pistas de una tarjeta de circuito impreso, **18**
Pixar, 5
pizza, 12
P_L, **122**
PLA *véase* arreglo de lógica programable
Plaza Sésamo, 146
PLD borrable eléctricamente, 349
PLD complejo (CPLD, complex PLD), 15, 16, 23, 872—881, 891
  tecnología de programación, 349
PLD de la serie X, 417
PLD secuencial, 343
PLD serie X, 707
PLD *véase* dispositivo de lógica programable
PMOS (MOS de canal-*p*), **87**
polaridad de salida, GAL, **344**
polarización, **39**
política, 3
ponerse a trabajar, 2, 194, 563, 573, 590, 617, 618, 623, 661, 693
port map, VHDL, 282, **283**
port, VHDL, 269, **270**
pos, palabra clave, 252
postulado, **195**
postulados de Huntington, **298**
potencia, 102
preajustable, 660, 905
preajuste (preset), **535**, **541**
precargado, **866**
precedencia
  de álgebra binaria, **197**, 203
  de operador ABEL, 251
  de operador VHDL, 287
precedencia de operador
  ABEL, 251
  VHDL, 287
prefijo binario, ABEL, **259**
prefijo hexadecimal, ABEL, **259**
prestar, 32, **43**, 43, 432

principio de dualidad, **203**
principio de superposición, **915**
prioridad, **377**
procedimiento, VHDL, **279**
procesador de palabras (texto), 10
proceso de corrida, VHDL, **290**
proceso de diseño de extremo, 266
proceso de diseño de extremo inicial, 265
proceso de VHDL suspendido, **290**
proceso, instrucción de, VHDL, **289**
proceso, VHDL, **289**
process, VHDL, **289**
producto canónico, **208**, 215, 220, 230
producto de sumas, 199
producto mínimo, **230**
producto parcial, **46**
producto velocidad-potencia, **139**
  TTL, 167
programa de generación de prueba, **903**
programa de software, 317
programabilidad en el sistema, **350**
programación, **318**
  CPLD, 349
  EEPROM, 843
  EPLD, 348
  EPROM, 842
  PROM, 842
  ROM de máscara, 841
programación de una EPROM, 842
programación versus diseño de máquina de estado, 564, 661
programador versus diseñador lógico, 313
programas, 325
programas de minimización, 236—244
PROM bipolar, 842
PROM *véase* memoria programable solo de lectura
propagación del acarreo, **434**
proporción de fallas, **909**, 911, 921
Prosser. F. P., 645
Protocolo de Internet (IP, Internet Protocol), 4
prueba, **902**
prueba de diagnóstico, **902**
prueba de pasa/no pasa, **902**
prueba de quemado, **910**
prueba del sistema, 164
prueba en el circuito, **904**
prueba, sistema de, 164
puerto JTAG, 350
puerto, VHDL, **270**
pulso de sincronía, **719**
punto binario, **27**
punto de base, **26**, 35
punto de multiplicación (·), **196**
punto decimal, 26
puntos de prueba, **904**
puntuación, 77
Purnell, Jeff, xxvi

Q

Quine, W. V., 236, 298, 299
Q versus QN, 536, 607, 650

R

Raaum, Dave, xxvi
RAID (redundant array of inexpensive disks, arreglo redundante de discos económicos), 67
Rambus DRAM (RDRAM), 891
RAM dinámica (DRAM), 854, 866—872
  sincrónica (SDRAM), 871
  temporización astuta, **869**
RAM estática (SRAM), **854**—861
  asincrónica, **861**
  célula, **856**
RAM ferroeléctrica, **854**
range, atributo, VHDL, **279**
range, VHDL, **273**
ranura temporal, **720**
RAS_L (row address strobe, selección de dirección de renglón), 868
rayos cósmicos, 58, 79
razón (cociente) de código, **78**
razonamiento circular, 569
RDRAM (Rambus DRAM), 891
realidades desagradables, 757
realización, **216**
realizar, 216
rebosamiento, **128**
rebote de contacto, **667**
rebote de terminal de tierra, 143
receptor de pulso, 618—624, 627
rectificador controlado de silicio (SCR, silicon-controlled rectifier), 112
  parasitario, 112
recursión, 243
red, **898**
Red Digital de Servicios Integrados (ISDN, Integrated Services Digital Network), 718
reducción de tabla de estado, 645
reflexiones, línea de transmisión, 125, 173, **915**
reg, palabra clave, ABEL, 628
región activa, **150**
región de saturación, 152
región no definida, 114, 154
registro, 671—674, 751
registro de corrimiento, **712**—733, 751
registro de corrimiento bidireccional, 714
registro de corrimiento con retroalimentación lineal (LFSR, linear feedback shift register), 603, 604
  contador, 730
registro de corrimiento de entrada en paralelo, salida en paralelo, **712**
registro de corrimiento de entrada en paralelo, salida en serie, **712**
registro de corrimiento de entrada serial, salida paralela, **712**
registro de corrimiento unidireccional, **714**
registro de corrimiento universal, **716**
registro de dirección de columna, **869**
registro de dirección de renglón, **869**
Regreso a cero bipolar (BPRZ, Bipolar Return-to-Zero), **72**
relación, ABEL, 258
relevador, 102
reloj, 7, **530**, 531
  comandado, 714
  distribución, 177
  en sistema sincrónico, 750
  fluctuación, inestabilidad de, 664
  frecuencia, **530**
  marca de, **530**, 550
  periodo, **530**

recuperación, 733
sesgo de, 663, 664, 750, **757**, 761
reloj controlado por compuerta, 714
representación de cubo, 237
representación de empaquetado BCD, **49**
representación de exceso B, **39**
representación de magnitud con signo, 50
representación de punto flotante, 848
representaciones de máquina de estado inconsistente, **661**
reproductor de disco compacto, 831
resistencia de Thévenin, **103**
resistencia de polarización directa, **147**
resistencia de saturación, 152
resistencia versus impedancia, 88
resistencia, polarización directa, 147
resistor
de arranque, **127**, 162, 163, 164, 385, 670
cálculo de, **133**
de desminución, 162, 164, 180, 906
resolución de problemas, 2
resta binaria, **32**
restablecimiento, 535, 900
máquina de estado, 565, 569
restablecimiento, máquina de estado, 570
resta de complemento a dos, **41**
resta de complemento a uno, 44
restador completo, **432**
restricciones, **267**
restricciones mecánicas, 901
resultado, VHDL, 276
retain, propiedad, ABEL, **675**
retardo, 244, 266, 267, **330**—337, 536, 758
máximo, 331, **333**
mínimo, 333, 336
separador de tres estados, 386
típico, 331, 333
TTL, 161
véase también retardo de propagación
retardo, circuito secuencial de retroalimentación, 624
retardo de propagación, 96, 138, 332, 336, 536, 536, 539, 540, 624, 677, 690, 691, 912
retardo de propagación t_p, 119
retardo delta, VHDL, **296**
retardo en el peor de los casos, **337**
retardo máximo, 331, 333
retardo mínimo, 331, 333, 336
retardo típico, 331, 333
Retorno a cero (RZ, Return-to-Zero), 71
retroalimentación externa, PLD, **691**
retroalimentación interna, PLD, **691**
return, VHDL, 276, 277
revisión del diseño, **896**
revolución, 5
revolución digital, 5
rieles, fuente de alimentación, **101**, 107
riesgo, **244**—248, 617, 750
dinámico, **247**, 750
en circuito secuencial de retroalimentación, 248
en diseño sincrónico, 248
estático, 674, 703, 750
estático-0, **245**
estático-1, **244**
función, **704**, 750, 784
fundamental, **624**, 677, 757
riesgo de 0 estático, **245**
riesgo de 1 estático, **244**
riesgo de función, **704**, 750, 784
riesgo de la temporización, 200
riesgo dinámico, **247**, 750
riesgo estático, 616, 674, 703, 750
riesgo fundamental, **624**, 677, 757
rizo, 422
Robbins, Tom, xxvi
Rogers, Fred, 534
rol, VHDL, 502
ROM bipolar, 840
ROM de máscara, **841**
ROM de máscara programable, **841**
ROM programable una vez, (OTP, one-time programmable), **843**
ror, VHDL, 502
Rudell, R. L., 300
ruido, 97, 124, 125, 163
conmutación, 143
ruido de conmutación, 143
ruptura del diodo, **147**
RZ (Return-to-Zero, Retorno a Cero), **71**

S

Salida
de colector abierto, 127, 328
de drenaje abierto, **127**, 129, 130
tolerante a 5 V, 174
salida auxiliar, PLD, **364**
salida de acarreo combinacional, 707
salida de acarreo con registros, 707
salida de colector abierto, 127, **165**, 328
salida de datos, **832**
salida de drenaje abierto, **127**, 129, 130, 328
salida de multiplexor, 402
salida de totem pole, TTL, **157**
salida de tres estados, **126**, 164, 402, 670
VHDL, 395
salida en cascada, circuito iterativo, **747**
salida en contrafase, TTL, **157**
salida flotante, 385
salida primaria, circuito iterativo, 747
salida registrada, 627
salida serial, registro de corrimiento, **712**
salida sobrecargada, TTL, 161, 162
salida tipo Mealy, 551, 558, 752, 796, 798
salida tipo Moore, 551, 556, 558, 632, 752, 798
salidas canalizadas, **552**
salidas constantes, PLA, 339
salidas de frontera, circuito iterativo, **421**, 747
salidas diferenciales, **177**
salidas en conflicto, 127, 132, **386**
salidas, en conflicto, 127, **132**
salidas, PLA, **337**
salidas tipo Moore, 797
salsa secreta, 881

SanDisk Corporation, 844
Sangiovanni-Vincentelli, A., 300
saturación, 175
saturado (ON), 152
SCR parásito, 112
SCR, *véase* rectificador controlado de silicio
Schmid, Hermann, 73
Schottky TTL de baja potencia (LS-TTL), 669
Schwarzenegger, Arnold, 534
SDRAM *véase* DRAM sincrónica
secciones opcionales, xxi, 38
secuencia de conteo pseudoaleatorio, 733
secuencia de longitud máxima, 730
secuencia de sincronización, **640**
segmento de arreglo, VHDL, **276**
segundo, s, 69
Seguridad Social, 644
Seidensticker, Robert B., 456
Seitz, Charles L., 785
select, instrucción, VHDL, 367
semáforos, 4
semáforos inteligentes, 809, 827
sensibilidad al tipo de letra (mayúsculas-minúsculas)
  ABEL, 249
  VHDL, 269
sensibilización de trayectoria, 300
sensor, 51
señal activa de reloj en estado alto, 530
señal activa de reloj en estado bajo, 530
señal de entrada asincrónica, **762**
señal de sincronía, **70**
señal en modo común, **177**
señal flotante, 385
señal *versus* variable, VHDL, 747
señales asincrónicas, 659, 750
separador de tres estados, **126**, **385**—391
separador ficticio, 605, 606, 609
separador, palabra clave, ABEL, **252**
separador, VHDL, **270**
serie 4000, **135**
sesgo, 177
sesgo de reloj, 750
sesgo de temporización, **626**
  salida, **741**
  separador de tres estados, 386
sesgo de temporización de salida, **741**
set, ABEL, 258, 629, 634
sexo, 902
Shamrock el perro, xxvi
Shannon, Claude E., 195, 298
signal, VHDL, 271
signo, 848
símbolo lógico, 356
  74x151 alternado, 473
  tradicional, 355
símbolos equivalentes, 317
símbolos lógicos tradicionales, 355
simulación, 11, **899**
  ineficiencia, **900**
simulación de transistor, 152
simulación versus PLD, 11
simulador, 10, **899**
  VHDL, **266**
simulador digital, 899
sincronizador, 2, 750, **762**
  falla del, **765**, 793
sincronizador de ciclo múltiple, 771
sincronizadores en cascada, **772**
sincrónico, 550
síndrome, **64**, 415
single stuck-at fault modelo de falla persistente simple, 263
sin restablecimiento, 565
síntesis, **266**, 297
  circuito combinacional, **193**
  máquina de estado, 591
  versus diseño, **194**
síntesis de circuito combinacional, **193**, 215—235
sistema de complemento de base, 35
sistema de exceso 2^{m-1}, **39**
sistema de magnitud con signo, **34**
sistema de números complementarios, 35
sistema disminuido de complemento de base, **38**
sistema numérico hexadecimal, **27**—29
sistema numérico octal, **27**—29
sistema numérico posicional, **26**
sistema telefónico, 4, 718, 831, 848
sistemas autotemporizados, **785**
sistemas sincrónicos, **659**, **750**—756
sla, VHDL, 502
SLDRAM (SyncLink DRAM), 891
sll, VHDL, 502
sobrecarga de operador, VHDL, **277**, 287
sobreexcitación, **904**
sobreimpulso, 173
SPICE, 899, 920
sra, VHDL, 502
SRAM asincrónica, **861**
SRAM sincrónica (SSRAM), **862**
  de escritura diferida con salidas canalizadas, **863**
  de escritura diferida con salidas de flujo continuo, **862**
  penalización de retorno, **864**
  retorno de bus cero (ZBT, zero-bus-turnaround), **864**
  ZBT con salidas canalizadas, **865**
  ZBT con salidas de flujo continuo, **864**
srl, VHDL, 502
SSI *véase* integración a pequeña escala
SSRAM de escritura diferida con salidas canalizadas, **863**
SSRAM de escritura diferida con salidas de flujo continuo, **862**
SSRAM de retorno de bus cero (ZBT zero-bus-turnaround), **864**
Stark, Brian, xxii
state, ABEL, **630**
state_diagram, ABEL, **629**
std_logic, tipo, VHDL, **273**, 396
std_logic_1164 VHDL, encapsulado, 281
std_logic_arith VHDL, encapsulado, 281
std_logic_signed VHDL, encapsulado, 281
std_logic_unsigned VHDL, encapsulado, 281
std_logic_vector, tipo, VHDL, 276
std_ulogic tipo, VHDL, 396
Steger, Jean-Pierre, xxvi
Stone, Harold S., xx, 298
Strunk, William. Jr., 455
subcubo, **58**
subcubo *m*, **58**
submáquina, **602**

subrayado, VHDL, 269
subtipo, VHDL, **273**, **396**
subtype, VHDL, **272**
suma binaria, **32**
suma canónica, **208**, 209, 215, 220, 223
suma completa, 227, 248, 309
suma de complemento a dos, **39**
suma de complemento a uno, **44**
suma de productos, 199
suma hexadecimal, **34**
suma lógica, 197
suma mínima, 225, 226, 227, 228, 229, 230
suma no redundante, 306
suma, operador de, 705
sumador, **430**
sumador binario en serie, **749**
sumador binario serial, 749
sumador completo, **431**, 432, 433
sumador de acarreo hacia delante, **435**
sumador de magnitud con signo, 35
sumador iterativo, 422, 432
sumador iterativo en grupo, **438**
sumador medio, **431**
sumadores, 430—443
sumar, 199, 212. 213, 432
Sunnyvale, California, 5, 808, 825
supresión de código cero, 72
sustractor, **430**
  completo, **432**
sustractor de magnitud con signo, **35**
sustraendo, 32
SyncLink DRAM (SLDRAM), 891
Synopsys, Inc., xxiv, 265

T

τ, constante de tiempo de resolución de metaestabilidad, **768**, 770
t_{AA}, **846**, **858**
T_0, **768**
tabla de aplicación J-K, 577
tabla de estado, **82**, 555, 564, **607**, 660
tabla de estado/salida, 553, 555, 560, 562
tabla de excitación, **573**
tabla de flujo, **613**
tabla de flujo primitiva, **618**, 618
tabla de función, **356**
tabla de implicante primo, **242**
tabla de resta en binario, **32**
tabla de salida, 560, 562
tabla de sumas en binario, **32**
tabla de temporización, **331**, 663
tabla de transición, **555**, 560, 562, 573, **606**
tabla de transición/excitación, **573**
tabla de verdad, 11, 20, **206**—209, 221, 236, 352, 356, 573, 832
  ABEL, 257, 260, 628
  notación, 352
tablero de alambrado impreso (PWB, printed-wiring board), **18**
t_{ACS}, **846**, **858**
t_{AH}, **859**
tambor rotatorio, 855
tarjeta de circuito impreso (PCB, printed-circuit board), **18**, 330, 758, 904
t_{AS}, **859**
Taylor, Douglas, 265
t_{CF}, **691**
t_{clk}, **766**
t_{CO}, **691**
t_{comb}, **766**
t_{CSW}, **859**
t_{DH}, **859**
t_{DS}, **859**
tecnología de circuito impreso (PCB) de línea fina, **18**
tecnología de montaje superficial (SMT, surface-mounted technology), **18**, 904. 906
telefonía digital, **718**
temperatura, 100, 101, 104, 331, 333, 608, 611, 770, 911
temporización, 330—332, 690—692
teorema, álgebra binaria, **198**
teorema de capa, **199**
teorema de combinación, **199**, 220, 239
teorema de consenso, **200**
  cubierta, **199**
  distribución de la multiplicación, 199
  dualidad, **203**
  entre paréntesis, 198
  expresión, **196**, 197
  ley distributiva, 199
  precedencia, **197**, 203
  teorema, **198**
  teorema de cubierta, **199**
  teorema de DeMorgan, **200**
  yuxtaposición, 197
teorema de DeMorgan, **200**, 485
  generalizado, 202, 204, 206, 212, 231, 374
teorema de implicante primo, **226**, 298
teorema generalizado de DeMorgan, **202**, 204, 206, 212, 231, 316, 374
teoremas de expansión de Shannon, generalizados, 304
teoremas generalizados de expansión de Shannon, **304**
tera- (T), 72
terminación, 102, 192, 918—920
terminación de fuente, **919**
terminación de Thévenin, **192**
terminación en serie, **919**
terminal activa en estado alto, 319
terminal activa en estado bajo, 319
terminal de E/S, PAL, **343**
terminal de salida, **13**, 353
terminal de salida de tres estados, 391
terminales bidireccionales, PLD, 342, 391, 682
terminales de E/S, 405
término de consenso, 617, 626, 674
término de suma, **207**
término del producto, **207**
término normal, **207**
término p de transición, **592**
término s de transición, **594**
términos del producto, PLA, **337**
test_vectors, ABEL, **251**, **262**
T (tera), 72
texto, **53**
t_H, **691**
The Phone Company (TPC), 4, 72, **73**, 718
tictac, 550, 551
tiempo de acceso desde la dirección, **846, 858**
tiempo de acceso desde la selección de chip, **846, 858**
tiempo de almacenamiento, **154**
tiempo de ascenso, (t_r), **114**

tiempo de ascenso, 125, 758
ECL, 180
tiempo de caída, 125, 758
ECL, 180
tiempo de caída (t_f), **114**
tiempo de configuración, 266, **540**, 541, 548, 549, 677, 692, 856
tiempo de configuración de datos, **859**
tiempo de deshabilitación de salida, **847**, **858**
tiempo de diseño, 23
tiempo de establecimiento de la dirección, **859**
tiempo de habilitación de salida, **847**, **858**
tiempo de instalación de datos, **859**
tiempo de recuperación, 537
tiempo de resolución de metaestabilidad, **766**
tiempo de retención, 266, **540**, 541, 548, 549, 677, 848, 856
tiempo de retención de dirección, **859**
tiempo de retención de salida, **847**, **858**
tiempo de simulación, VHDL, **296**
tiempo de transición, 96, **114**, 125
tiempo medio entre fallas (MTBF, mean time between failures), 768, 769, 771, 772, 773, 911, 912
tiempo muerto, 386
Tin Toy, 5
tipo arreglo no restringido, VHDL, **276**
tipo de alambre, **898**
tipo de arreglo, sin restricción, **276**
tipo de CI, **328**
tipo de componente, 897
tipo definido por el usuario, VHDL, **272**
tipo enumerado, VHDL, **272**, 815
tipo "flash", 843
tipo sin resolver, VHDL, **396**
tipo, VHDL, **271**, 272
no resuelto, **396**
tipos predefinidos, VHDL, **271**
Tison, Pierre, 299
title, instrucción, **250**
TL7705, 565
to, VHDL, **273**, 276
t_{OE}, **847**, **858**
t_{OH}, **847**, **858**
Tortugas Ninja Adolescentes Mutantes, 187
t_{OZ}, **847**, **858**
TPC *véase* The Phone Company
t_{PD}, **690**
t_{pHL}, **120**
t_{pHZ}, 386
t_{pLH}, **120**
t_{pLZ}, 386
t_{pZH}, 386
t_{pZL}, 386
t_r, **766**
Tracy, Ted, xxvi
traductor de nivel, 175
transceptor, **390**, 392
transceptor de bus, **390**, 392
transceptor de fibra óptica, 176
transistor, 836
bipolar de unión, **84**, **149**—152
MOS, **85**
MOS de canal *n* (NMOS), **87**
MOS de canal *p* (PMOS), 87
Schottky de bloqueo, **154**
transistor bipolar de unión (BJT, bipolar junction transistor), **84**, **149**—152
transistor con emisor múltiple, 156, 164
transistor de efecto de campo con semiconductor de óxido y metal (MOSFET, metal-oxide semiconductor field-effect transistor), **85**—88
transistor MOS de canal *n* (NMOS, *n*-channel MOS), **87**
transistor MOS de canal-*p* (PMOS *p*-channel MOS), **87**
transistor MOS de compuerta flotante, **348**, **842**
transistor *pnp*, **150**
transistor Schottky, **154**
transitorio de bajo voltaje, 173, **917**
trayectoria de la señal, 119
trayectoria de retardo, 663
trayectoria furtiva, **838**
trayectoria, señal, **119**
trayectorias críticas de señal, **901**
trazado de PCB, 895, 897, 898
trazado de un diagrama esquemático, 214
trucos, 761
TRUE, 3
true, VHDL, 272
truth_table, ABEL, 257
t_{setup}, 766, 766
t_{SU}, **691**
TTL, **158**
TTL, **160**
TTL, 161
TTL, **161**
TTL, 161
TTL, 168
TTL *véase* lógica transistor-transistor
TTL/CMOS, interfaz, 102, 105, 136
tubos, 146
tubos de vacío, 146
Tuinenga, Paul W., 920
TV, 529, 530
two-dimensional código bidimensional, **66**
t_{WP}, **859**
type, VHDL, 272

U

ubicación de los componentes en la tarjeta de circuito impreso, 199
ubicación del término del producto, **875**
ULA Unidad de Lógica Aritmética (ALU Aritmetic Logic Unit), 398, 430
umbral, 8
unclocked assignment operador de asignación no temporizado, =, 252, 254
unidad central de procesamiento (CPU, central processing unit), 831
unidad de control, 750
unidad de datos, **750**
unidad de lógica y aritmética (ULA/ALU), **439**
unión *pn*, **145**, 150
use, cláusula, VHDL, **280**
U.S. patentes, 74

V

valor de estado, ABEL, **630**
valor del componente, 897
valor lógico, 80
  CMOS indefinido, 86
valor lógico indefinido, CMOS, 86
variable de estado, **530**
variable versus señal, VHDL, **747**
variable, VHDL, **271**, **289**, 290
variables de estado de corriente, ABEL, **634**
variables de estado siguiente, ABEL, **634**
V_{CC}, 90
VCR, 529
V_{DD}, 90
Véase también dispositivos GAL
vector de estado, ABEL, **629**
vectores de prueba, **351**, **903**
  ABEL, **251**, 638
Veitch, E. W., 298
velocidad, 5, 219
  circuito combinacional, 217, 219
  CMOS, 96, 113
  PAL, 344
velocidad de código, 78
ventana de decisión, **767**
velocidad de la luz, 912
verificación, **266**
verificación de redundancia cíclica (CRC cyclic-redundancy-check)
verificación de suma (checksum), **68**
verificación de temporización, **266**, 267, **899**
verificación funcional, **266**, 267
verificador de las reglas de diseño, **899**
verificador de temporización, 10, **899**
Verilog, **265**
Verilog HDL *véase* Verilog
VHC (Very High-speed CMOS, CMOS de muy alta velocidad), **137**, 173
VHCT (Very High-speed CMOS, CMOS de muy alta velocidad), (TTL compatible), **137**
VHDL, **265**
VHDL, 9, 14, 16, **264**, **265**, 921
  /= operador, **287**
  :operador, **290**
  < operador, **287**
  <= operador, **287**
  = operador, **287**, **427**
  > operador, **287**
  >= operador, **287**
  after, instrucción, **295**
  architecture, instrucción, 269
  argumentos, **276**
  arquitectura, **268**
  array, instrucción, **274**
  arreglo, **274**
  arreglo, literal de, **275**
  attribute, instrucción, 816
  bejín, instrucción, 269
  bit, tipo, 271
  bit_vector, tipo, 271
  boolean, tipo, 271, **272**
  buffer, instrucción, **270**
  case, instrucción, **292**
  comentarios, **269**
  compilador, **265**
  componente, declaración de, **283**
  component, instrucción, **282**, 283
  condicional, **287**
  constant, instrucción, **272**
  constante, **274**
  conversión de tipo explícito, **510**
  character tipo, 271, 272
  declaración constante, **274**
  definición de arquitectura, **269**, **270**
  descripción de comportamiento, **289**
  descripción de flujo de datos, **286**
  diseño de comportamiento, **289**
  diseño de flujo de datos, 286
  downto, 273, 276
  else, instrucción, 269, **287**, **292**
  elsif, instrucción, **292**
  end, instrucción, 269, **270**
  entidad, **268**
  entity, declaración, **269**
  entity, instrucción, 269, **270**
  event, atributo, **643**, **678**
  exit, instrucción, **293**
  false, 272
  flujo de diseño, 264
  for, ciclo, **293**
  índice de arreglo, **274**
  instrucción concurrente, **282**
  instrucción de asignación de señal concurrente, **286**
  operador de concatenación &, 276, 502, 743
  operador de igualdad, (=), 427
  parámetros reales, **276**
  retardo delta, **296**
  sección de arreglo, **276**
  tipo enumerado, 272, 815
  tipos de arreglo, **274**
VHDL-87, **264**, 300
VHDL-93, **264**, 300
VHSIC, 264
$V_{IHmáx}$, TTL, **161**
$V_{IHmín}$, 100, **141**
$V_{IHmín}$, TTL, **158**
$V_{ILmáx}$, 100, **140**
$V_{ILmáx}$, TTL, **160**
VLSI personalizado, 348
VLSI *véase* integración a muy grande escala
$V_{OHmín}$, **100**
$V_{OHmín}$ TTL, **158**
$V_{OHmínC}$, **142**
$V_{OHmínT}$, **142**
$V_{OLmáx}$, 100
$V_{OLmáx}$, TTL, **160**
$V_{OLmáxC}$, **142**
$V_{OLmáxT}$, **142**
voltaje de la fuente de alimentación, 100, 101, 171, 331, 333, 608, 611, 770
voltaje de Thévenín, **103**
voltaje, fuente de alimentación de, 333, 608, 611, 770
Vulcan, 194

W

wait instrucción, VHDL, **295**
Wakerly, Carmela F. xix
Wakerly, John F. xix, 73, 74
Wakerly, Kate xxvii
Wakerly, Ralph F. iv
Waser, Shlomo, 73
WE_L (write enable, habilitación de escritura), **870**

Weldon, E. J. Jr., 73
Weste, Neil H. E., 183
WHEN, instrucción anidada, ABEL, 254
WHEN, instrucción, ABEL, **253**
when, VHDL, 269, **287**, **292**
while, ciclo, VHDL, **295**
White, E. B., 455
Wickes. William E., 183
Widdoes. L. Curtis, xxvi
Widmer, Albert, 74
Winkel, D. E., 645
WITH, ABEL, **636**
Wood, Samuel F., xxvi
work, biblioteca, VHDL, **280**, 522
www.ddpp.com, xxi

X

Xilinx, Inc., xxiv, 16
Xilinx University Program, xxiv
XNOR, compuerta, 426, 698, 705
XOR, compuerta, **304**, 306
como comparador, 420
XOR, estructura, 707
XOR, operación, 305

Y

Young, Des, xxvi
yuxtaposición, 197

Z

ZBT SSRAM con salidas canalizadas, **865**
ZBT SSRAM con salidas de flujo continuo, **864**
zener, diodo, **147**
Zucker, Irwin, xxvi

PROGRAMAS EDUCATIVOS, S.A. DE C.V.
CALZ. CHABACANO NO. 65,
COL. ASTURIAS, DELG. CUAUHTEMOC,
C.P.06850, MÉXICO, D.F.

EMPRESA CERTIFICADA POR EL
INSTITUTO MEXICANO DE NORMALIZACIÓN
Y CERTIFICACIÓN A.C. BAJO LAS NORMAS
ISO-9002:1994/NMX-CC-004:1995
CON EL NO. DE REGISTRO RSC-048
E ISO-14001:1996/NMX-SAA-001:1998 IMNC/
CON EL NO. DE REGISTRO RSAA-003

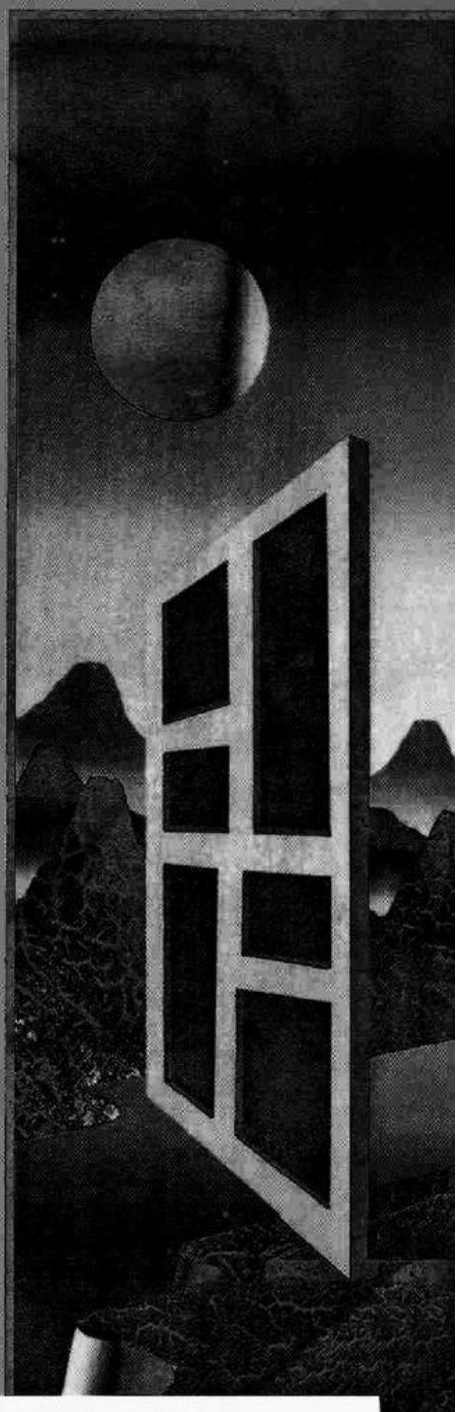
Esta nueva edición completamente revisada conjunta la precisión académica y la experiencia práctica, en una introducción formal de los principios básicos del diseño digital y los requerimientos de los sistemas a nivel tarjeta de circuito impreso y VLSI.

Con más de 22 años de experiencia en los ámbitos industrial y académico, John Wakerly ha enseñado en forma directa a miles de estudiantes de ingeniería y de forma indirecta a decenas de miles de estudiantes a través de sus textos universitarios. También ha diseñado directamente sistemas digitales reales que representan muchos millones de dólares en ganancias.

Este libro cubre los principales bloques de construcción del diseño digital mediante varios niveles de abstracción, desde las compuertas CMOS hasta los lenguajes de diseño de hardware. Las funciones importantes como son compuertas, decodificadores, multiplexores, flip-flops, registros y contadores se analizan en cada nivel.

Entre las nuevas características de esta edición se incluyen un menor énfasis en los procedimientos manuales y en el diseño de dispositivos PLD y FPGA; y lenguajes de diseño de hardware para obtener la mayor ventaja de los modernos componentes y herramientas de software. El ánalisis de los lenguajes de descripción de hardware (HDL) incluye los lenguajes VHDL y ABEL.

Visite el sitio Web de este libro en:
www.pearsoneducacion.net/wakerly
donde encontrará actividades
y apoyos para el profesor.



ISBN 970-17-0404-5
90000
9 789701 704042

PEARSON
Educación ®

Visítenos en:
www.pearsoneducacion.net

Created with a trial version of RcaJPEG
registered version doesn't display this notice